NRL Report 9296

Creation, Validation, Testing, and Data Management of a Knowledge Base Designed for a Technician's Assister System for the AN/SQS-53B, Unit 26, Using a Fault Isolation System Shell

JOSEPH A. MOLNAR

Communications System Branch
Information Technology Division

December 17, 1990

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 17, 1990	Interim

4. TITLE AND SUBTITLE
Creation, Validation, Testing, and Data Management of a Knowledge Base Designed for a Technician's Assister System for the AN/SQS-53B, Unit 26, Using a Fault Isolation System Shell

5. FUNDING NUMBERS
PE-25620N
TA-
WU-L .57-139

6. AUTHOR(S)
Molnar, J.A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Research Laboratory
Washington, DC 20375-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NRL Report 9296

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Oceanographic & Atmospheric Research Lab
Stennis Space Center, MS 39529-5004

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

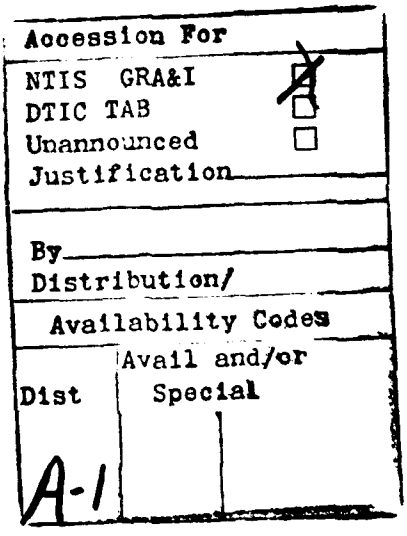
13. ABSTRACT *(Maximum 200 words)*

A system using the Fault Isolation System (FIS) shell was designed for Unit 26 of the SQS-53B Sonar System. The sonar system unit has a mixture of analog and digital components. Over 100 replaceable modules are in Unit 26. The knowledge base created to describe the circuit topology of Unit 26 contains over 3,000 rules and was ten times larger than any other system modeled for FIS. A methodology is described for creation of large knowledge bases and of the data management requirements. A format of the data management structure is presented in this report. Software written in C for data manipulation is described in this report. The diagnostic, compilation, verification, and validation process for the creation of a viable FIS knowledge base is also presented in this report. Knowledge base manipulation software was written in LISP or in C interfaced to LISP on a SUN 3/110. Validation was performed at the Naval Underwater Systems Center, New London, CT.

14. SUBJECT TERMS		15. NUMBER OF PAGES
Knowledge engineering	Verification	151
Expert systems	Validation	16. PRICE CODE
Artificial intelligence	Maintenance aid	

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

CONTENTS



CREATION, VALIDATION, TESTING AND DATA MANAGEMENT OF A KNOWLEDGE BASE DESIGNED FOR A TECHNICIAN'S ASSISTER SYSTEM FOR THE AN/SQS-53B, UNIT 26, USING A FAULT ISOLATION SYSTEM

INTRODUCTION

FIS (Fault Isolation System) is a model-based expert system shell developed to aid in the diagnostics of systems containing analog components. The shell written in LISP contains numerous functions that calculate the diagnostic testing sequence based on probabilistic algorithms. The shell acts on a knowledge database developed specifically for each unit under test (UUT). This report outlines the process required for the formation of the knowledge database.

The largest knowledge database, to date, has been for Unit 26, the CP Processor of the AN/SQS-53B sonar system. This unit consists of 12 channels that process the left, center, and right beams of the sonar system. A mixture exists of analog and digital functions, with over 100 replaceable modules in the unit. A knowledge engineer constructed rules for the system from the schematic diagrams after considering the designed test points to be the only real test points in the system. The result was a database containing over 3000 rules and 600 tests with test instructions that describe the operation of Unit 26.

During the development of the database for the whole system, a determination was made that by creating a hierarchy of knowledge databases, based on the system functionality, the fault isolation would improve. The resulting hierarchy was named the Technician's Assister System (TAS). During fault isolation, TAS loads individual knowledge databases for each functional area. This effort resulted in 12 distinct knowledge databases that describe the 12 functional areas of the system. Each knowledge database required the same construction techniques. FIS includes several software utilities for developing a knowledge database; they are all incorporated within the FIS editor. The editor facilitates the creation of a syntactically correct knowledge database, however its sequential structure of data entry is time consuming. Therefore, enhancements were made to the FIS editor to permit transformation of data structures from a data file format to structures that are syntactically correct for FIS. The enhanced version of FIS used in TAS is referred to as TA/FIS. Also, several auxiliary programs were created to aid in the process. This report focuses on techniques used to make the transformations from the standard data structures. The data structures and the software developed are described in this report.

The process used in developing a viable knowledge database is knowledge acquisition, data structuring, compilation, and validation. Several methods used for creating the FIS knowledge database are described in this report. The appropriateness of each method is discussed. The compilation and validation processes are presented to instruct users on all processes required to obtain a functioning FIS knowledge database. The techniques used for each are described, as well as the types of errors and methods required to resolve them.

Finally, the knowledge database is not a static entity. Through the service lifetime of a naval warfare system such as the AN/SQS-53B, numerous changes are made to the hardware to rectify deficiencies or enhance performance. An AI system used in the maintenance of such systems must be continually updated to remain a viable entity for system maintenance. Thus, this report also provides information on how the available techniques, contained in FIS or subsequent FIS enhancements, can be used in configuration management of the knowledge database. The report provides the user, program managers, and defense contractors with information on how to maintain FIS knowledge databases within the structure of a changing warfare system configuration.

Manuscript approved September 5, 1990.

KNOWLEDGE DATABASE

Three stages exist in the formation of a UUT knowledge database. The first stage consists of performing knowledge acquisition. For Unit 26 of the AN/SQS-53B this entails using schematics. (Future approaches may use information provided by computer aided engineering workstations that require minimal human intervention in the actual process of knowledge acquisition.) The intended result is to form a database that facilitates the validation and configuration management. This is an important requirement because fielded systems regularly require engineering changes or entire upgrades to correct system deficiencies or problems. Therefore, the original knowledge databases would require alterations to reflect changes in system software and hardware. The use of a database structure is an added feature in the development performed for the AN/SQS-53B application. Previous methods used only the structure that resulted from direct entry of data by using the FIS editor.

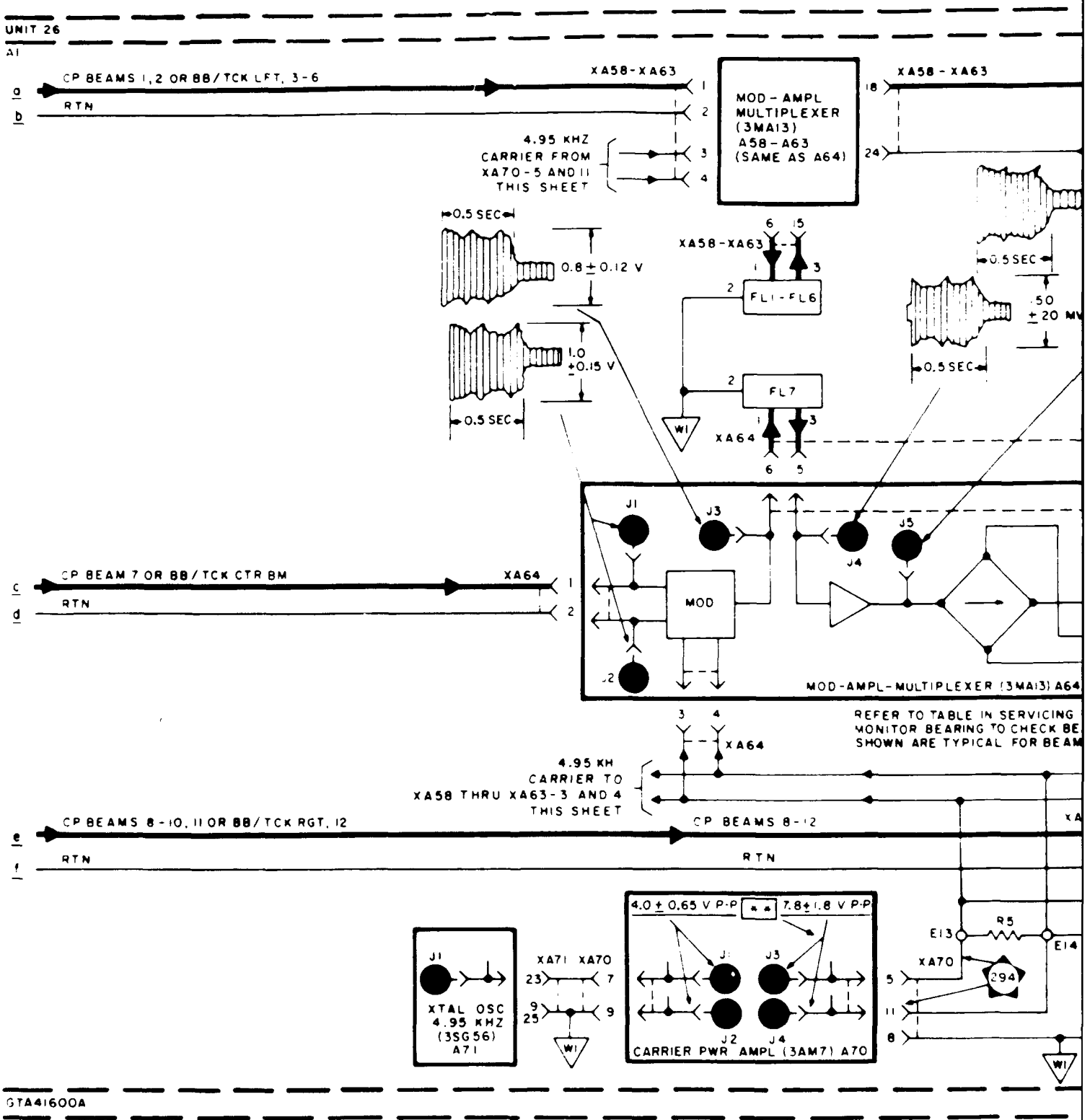
The second stage involves conversion of the initial knowledge database structure into a similar structure compatible with consumption by a LISP program. At this stage, units of "like" information are grouped into LISP lists. Primarily there are five forms of data: rules, tests, preconditions, orders, and instructions. Other forms of information related to graphics presentation may also be included. In this sonar unit application, the graphics information was unnecessary. Reference to the creation of knowledge database graphics information is available elsewhere [1]. All five data types used by the FIS sonar system application require a unified data format. The FIS format requires that all of the information in the separate data structures be grouped to form the knowledge database. To obtain a functioning knowledge database requires several steps; they are conversion, editing, error checking, compilation, and structure addition.

The third stage involves using a LISP function to add information to the FIS knowledge database and to properly structure the file for input into the FIS compiler. The result of these steps is a single file with a *.lisp* suffix.

Knowledge Acquisition

Knowledge acquisition is the first step in preparing a FIS knowledge database. The two types of information acquired during this stage are causal information and auxiliary information. Causal information describes the cause-effect relationship of the modules. Auxiliary information describes information related to a specific condition, state, or test. All the information is obtained by a knowledge engineer. The process examines a schematic and constructs the information in a compatible format. Figure 1 shows an example of the schematic that is an excerpt of the schematics for the whole unit. Several things should be noted in this figure; first, the large block located roughly at the right-hand side near the center of the figure is labeled *Multiplex Gates (3LA23) A16*. This is an example of what is later referred to as a module. The terminals for this module are labeled with numbers. The numbered terminals are normal input or outputs from the module and are not generally available to technicians for testing purposes, but for purposes of FIS these terminals are useful in defining rule relationships. However they are not useful as test points. Test points, on the right-hand side of the figure, are also defined with large round dots such as the one labeled *A11J7*. The others can also be readily seen. Information on the type of measurement and the expected value for the test appears next to the test point. Another type of test point that is also displayed is a Performance Monitoring Fault Locator (PMFL) test point, shown roughly in the center of the figure. It appears as a pentagon with the number 297 inside. This type of test point is used in the TAS system hierarchy to provide functional isolation. Also, the lines connecting the modules have arrowheads to indicate the direction of signal flow. From this information terminals are able to be defined as input or output terminals.

When collecting information for the knowledge database, most items are available from the schematics or elsewhere within the DATOM [2] for the sonar system. The knowledge engineer must extract all of the useful information and organize it into rules, tests, instructions, preconditions or orders, as appropriate. The format for these items is described in the following sections. All of the information must eventually be available in computer files. In the development of the TA/FIS for the AN/SQS-53B an ASCII text file format was created and used. These formats are described in the text, and examples are available in the appendices. Other formats could also be created. Currently, all of the information used in the sonar system application has been converted to a database format to



CP Signal Processing Channel

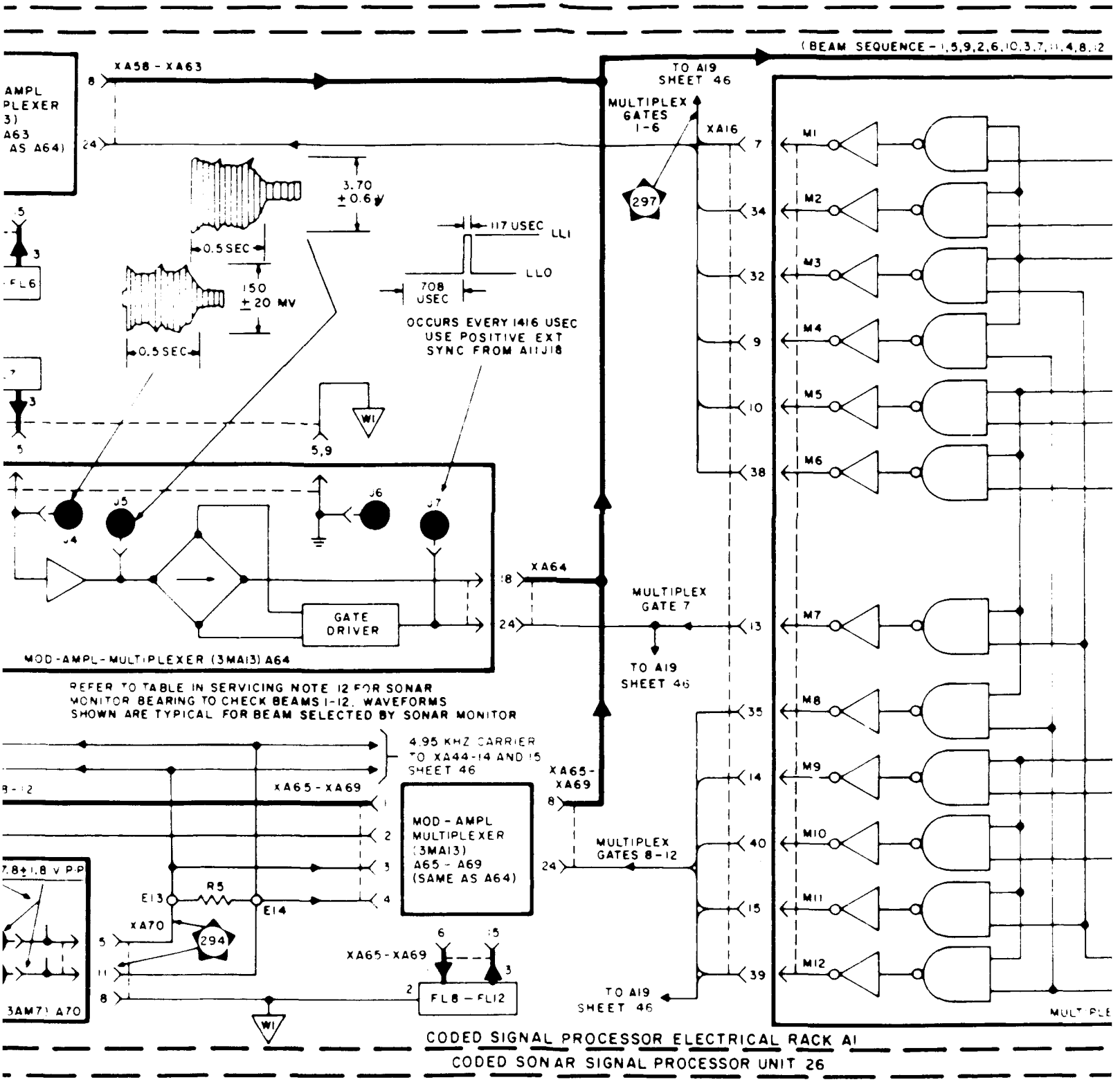
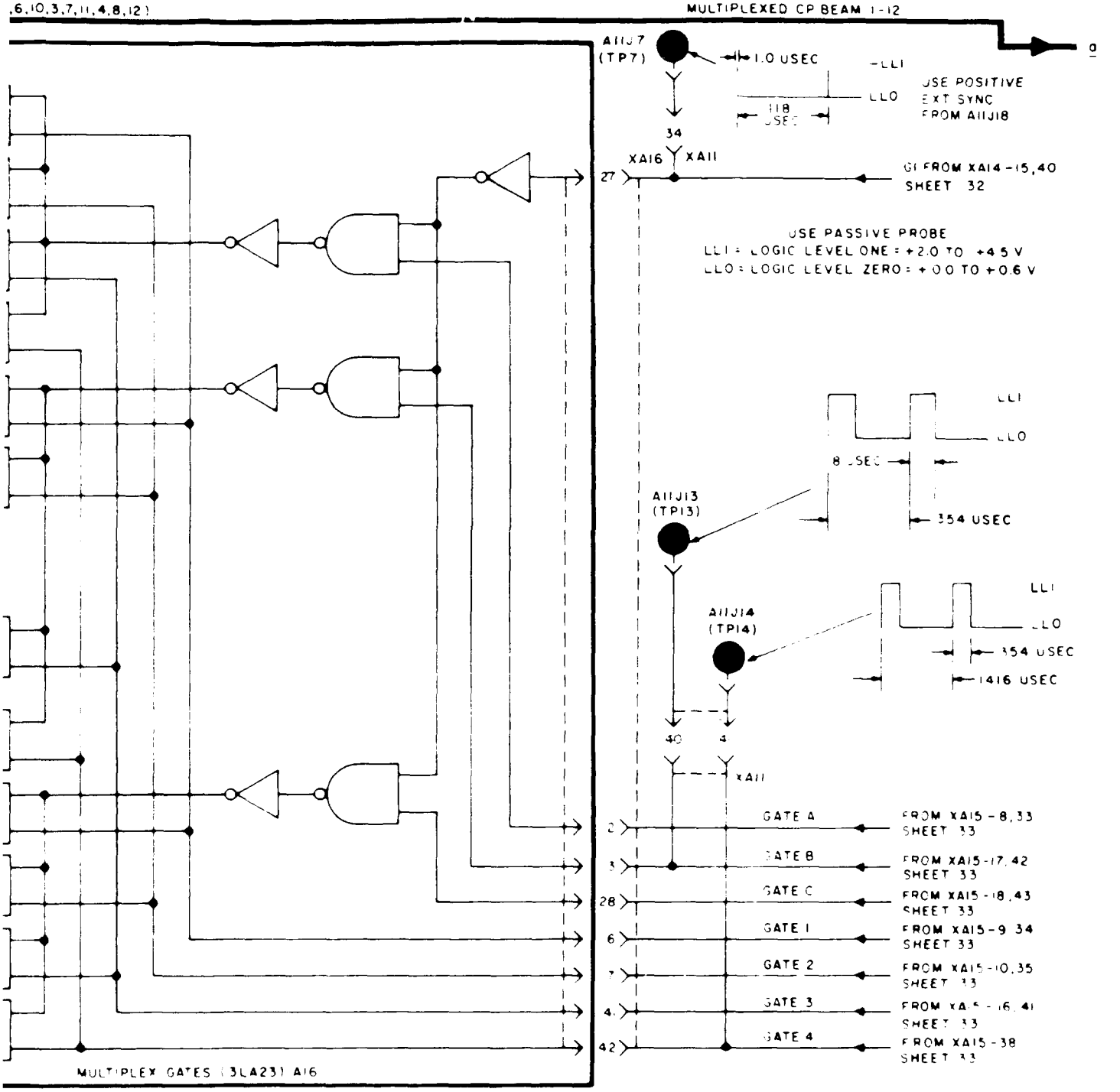


Fig. 1 - Schematic diagram for Unit 26 of the AN/SQS-53B displays general interconnection relationships (pri
placed on module *A16*) [2]



1 relationships (primary focus is

facilitate data management. The current format and the database program used in data management will be described in a later publication. The format described here is presented as a viable example of an implemented format and contains the essence of the information that is required by any format. Numerous references will be made to Fig.1 to exemplify the process of knowledge acquisition.

Causal Rule File

During fault isolation, the assumption is that the system is acting irregularly and/or that a fault indicator is present and indicates a system malfunction. Given these conditions, the human performing the troubleshooting will use the technician's assister version of FIS (TA/FIS) to isolate the fault. TA/FIS will indicate which replaceable units (modules) have failed based on the test results of a fault isolation session that consists of the computer providing suggestions on the most efficient tests to be made by the technician. The session starts with the assumption that a module has failed, resulting in an abnormality in the system. This form of causal reasoning provides the basis for the contents of the causal rule file.

One of two states can be present to incur a fault indication at the module output. One state results when an input to the module is faulty, thus indicating a failure upstream of the tested module. The other state occurs when the module itself is bad. An upstream module is a module that connects to the test point, but occurs in the electrical signal flow before the test point. In Fig.1, following the signal flow from terminal 13 of module *A16*, it is evident that the module *A16* is upstream of the *MOD-AMPL-MULTIPLEXER (3MA13) A64* module. Downstream is then defined as all modules whose test results rely upon the integrity of the modules before it in the signal path. Conversely, module *A64* is downstream of module *A16*.

An example of the first state occurs when a poor signal enters module *A16* and propagates through the module to output terminal 13. In this case module *A16* is not faulted but it presents a fault at its output, thus implicating it as a possible fault. To exemplify the second state, if module *A16* was faulted in such a way that results in a poor signal from terminal 13, its effect propagates downstream to module *A64*. In this case module *A16* is directly responsible for the presentation of a faulty signal at terminal 13. The causal rules define this relationship as a given module failure, or cause that produces an effect at an output terminal, provided the conditions of the UUT test are consistent. The causal rule is the statement of the cause and effect relationship that exists between interconnected modules in an electronic circuit.

The different individual components of the causal rule file are now defined along with the format. The essential format elements of the causal rule file are the module name, a rule number (for accountability purposes), a cause, an effect, a type (optional), and a precondition. Other optional items included are titles, headers and footers, dates, headings, and descriptive comments. An example of the format used for the causal rule file is included in Appendix A.

Module Name

The format for the module name, within the rule database, must be descriptive and distinctive for the UUT. The designation of a module may occur anywhere within the file, but must be before the first rule for that module. It may be of any length, but must have a letter, between A and Z, as its first character. The format defines the module name by placing the word module and a colon followed by one or more spaces before the actual module name, as indicated by the example in Fig.1.

Module: A26A1A16

In this example, and for most module names used for the AN/SQS-53B Unit 26, the leading A acts as an alphabetic place holder. The 26 following the leading A is indicative of the subsystem unit, as seen at the bottom center of Fig.1. The next two characters, A1, designate the electrical rack location of the module, seen also at the bottom center of Fig. 1, just above the unit designation. The final characters, A and a group of numbers (in this case 16), indicate the specific module within the unit as noted in the figure.

No other characters may occur on the line in which the module name occurs unless it is a comment and adheres to the comment format.

Rule Number

Figure 2 shows that each rule consists of a cause, an effect, type (optional), and precondition. A number, as the first character on a line, denotes that a rule follows. One or more spaces separate the number from the components of the rule. The number acts to order the rules of each module, so that humans may account for their presence or absence. A rule, once created, maintains a numbered place in the rule list. The user removes the rule that is no longer appropriate, but the rule number should remain to mark its place. The rule has no relation to the schematics of the sonar system; it is purely a construction of knowledge database creation.

In management of the database the number provides a place for the knowledge engineer to account for deletions of rules from previous versions by allowing space for comments relating to the deletion. Additional information, if needed, resides in the comments that follow the modules rule set. A descriptive comment such as the date and appropriate rationale should replace deleted rules. The rule number allows the creators of the knowledge database to identify rules within a unit. The number also announces that the remainder of the line is a rule. The software converts the database to a format compatible with FIS and uses this convention to identify rules. Numbers may also have comments that follow and conform to the format for comments. Other extraneous items that follow a comment either produce an immediate error when converted, or are processed, thus producing an undefined rule for the knowledge database. Undefined rules produce syntax errors during compilation.

Rules

Rules consist of elements such as cause, effect, type (optional) and precondition. The following sections describe each of these elements.

Several strategies are used in the creation of the rule database. The first strategy uses a point-to-point internal module network structure. This means that each input point of a module connects directly to the module output point through the use of a logical construct. Figure 2 (rules 1-36) shows this first strategy. This strategy uses the input terminal of the module being examined or the output terminal of the module immediately upstream to define the cause. Either terminal is allowable in this definition, provided other rules maintain the logical connection. A pseudonode allows the use of both the input and upstream output terminal by providing the logical interconnection between them. A pseudonode is a node that has physical existence over a connection length, not at a single point, as for real terminals.

Out of necessity for pseudonodes, virtual modules were created; modules that have ambiguous boundaries in the physical system but that are necessary because of a requirement to assure system connectivity. Often virtual modules describe physical entities such as backplanes that have no definite terminals or boundaries. Each interconnection then becomes part of a virtual module. This increases the total number of rules, but has the positive effect of defining all interconnections.

Beside upstream faults producing effects at output terminal, the module can produce a faulty output, as Fig. 2 shows in rules 37 to 48. The knowledge database requires that for every module at least one rule must be present that identifies the module as the producing source of improper effect at an output terminal. A warning occurs during compilation to indicate an error in knowledge database syntax.

A second strategy, similar to the first, attempts to reduce the number of rules. This strategy uses pseudoterminals, terminals that are not physically present but are logical creations that possess a central location between any two physical terminals, input and output, within a module. Rules 61 to 80 show this type of strategy, since the terminal *a26a1a16* is a logical creation as is the parameter *mplxr_gates*. These are extremely useful in describing highly parallel systems, because they allow the knowledge engineer to describe the physical system in fewer rules. In the example of module *A16*, four gates exist that control the path of 12 separate signals. The use of the pseudoterminal reduces the number of rules from 48 to 16, rules 73, 75, 77, and 79 do not relate directly to the

Module: a26a1a16_mplx_gates

No	Cause	Effect	Type	Precondition
1	a26a1a11J7 waveform bad	a26a1a58J7 gate_select bad	s	t
2	a26a1a11J7 waveform bad	a26a1a59J7 gate_select bad	s	t
3	a26a1a11J7 waveform bad	a26a1a60J7 gate_select bad	s	t
4	a26a1a11J7 waveform bad	a26a1a61J7 gate_select bad	s	t
5	a26a1a11J7 waveform bad	a26a1a62J7 gate_select bad	s	t
6	a26a1a11J7 waveform bad	a26a1a63J7 gate_select bad	s	t
7	a26a1a11J7 waveform bad	a26a1a64J7 gate_select bad	s	t
8	a26a1a11J7 waveform bad	a26a1a65J7 gate_select bad	s	t
9	a26a1a11J7 waveform bad	a26a1a66J7 gate_select bad	s	t
10	a26a1a11J7 waveform bad	a26a1a67J7 gate_select bad	s	t
11	a26a1a11J7 waveform bad	a26a1a68J7 gate_select bad	s	t
12	a26a1a11J7 waveform bad	a26a1a69J7 gate_select bad	s	t
13	a26a1a16-2 gate_a bad	a26a1a58J7 gate_select bad	s	t
14	a26a1a16-2 gate_a bad	a26a1a59J7 gate_select bad	s	t
15	a26a1a16-2 gate_a bad	a26a1a60J7 gate_select bad	s	t
16	a26a1a16-2 gate_a bad	a26a1a61J7 gate_select bad	s	t
17	a26a1a11J13 gate_b bad	a26a1a62J7 gate_select bad	s	t
18	a26a1a11J13 gate_b bad	a26a1a63J7 gate_select bad	s	t
19	a26a1a11J13 gate_b bad	a26a1a64J7 gate_select bad	s	t
20	a26a1a11J13 gate_b bad	a26a1a65J7 gate_select bad	s	t
21	a26a1a16-28 gate_c bad	a26a1a66J7 gate_select bad	s	t
22	a26a1a16-28 gate_c bad	a26a1a67J7 gate_select bad	s	t
23	a26a1a16-28 gate_c bad	a26a1a68J7 gate_select bad	s	t
24	a26a1a16-28 gate_c bad	a26a1a69J7 gate_select bad	s	t
25	a26a1a16-16 gate_1 bad	a26a1a58J7 gate_select bad	s	t
26	a26a1a16-16 gate_1 bad	a26a1a62J7 gate_select bad	s	t
27	a26a1a16-16 gate_1 bad	a26a1a66J7 gate_select bad	s	t
28	a26a1a16-17 gate_2 bad	a26a1a59J7 gate_select bad	s	t
29	a26a1a16-17 gate_2 bad	a26a1a63J7 gate_select bad	s	t
30	a26a1a16-17 gate_2 bad	a26a1a67J7 gate_select bad	s	t
31	a26a1a16-41 gate_3 bad	a26a1a60J7 gate_select bad	s	t
32	a26a1a16-41 gate_3 bad	a26a1a64J7 gate_select bad	s	t
33	a26a1a16-41 gate_3 bad	a26a1a68J7 gate_select bad	s	t
34	a26a1a11J14 gate_4 bad	a26a1a61J7 gate_select bad	s	t
35	a26a1a11J14 gate_4 bad	a26a1a65J7 gate_select bad	s	t
36	a26a1a11J14 gate_4 bad	a26a1a69J7 gate_select bad	s	t
37	a26a1a16_mplx_gates	a26a1a58J7 gate_select bad	s	t
38	a26a1a16_mplx_gates	a26a1a59J7 gate_select bad	s	t
39	a26a1a16_mplx_gates	a26a1a60J7 gate_select bad	s	t
40	a26a1a16_mplx_gates	a26a1a61J7 gate_select bad	s	t
41	a26a1a16_mplx_gates	a26a1a62J7 gate_select bad	s	t
42	a26a1a16_mplx_gates	a26a1a63J7 gate_select bad	s	t
43	a26a1a16_mplx_gates	a26a1a64J7 gate_select bad	s	t
44	a26a1a16_mplx_gates	a26a1a65J7 gate_select bad	s	t
45	a26a1a16_mplx_gates	a26a1a66J7 gate_select bad	s	t
46	a26a1a16_mplx_gates	a26a1a67J7 gate_select bad	s	t

Fig. 2 - Sample list of rules for module *A16*

47 a26a1a16_mplx_gates	a26a1a68J7 gate_select bad	s	t
48 a26a1a16_mplx_gates	a26a1a69J7 gate_select bad	s	t
49 a26a1a77J4 volts bad [4.5v supply]	a26a1a58J7 gate_select bad	s	t
50 a26a1a77J4 volts bad	a26a1a59J7 gate_select bad	s	t
51 a26a1a77J4 volts bad	a26a1a60J7 gate_select bad	s	t
52 a26a1a77J4 volts bad	a26a1a61J7 gate_select bad	s	t
53 a26a1a77J4 volts bad	a26a1a62J7 gate_select bad	s	t
54 a26a1a77J4 volts bad	a26a1a63J7 gate_select bad	s	t
55 a26a1a77J4 volts bad	a26a1a64J7 gate_select bad	s	t
56 a26a1a77J4 volts bad	a26a1a65J7 gate_select bad	s	t
57 a26a1a77J4 volts bad	a26a1a66J7 gate_select bad	s	t
58 a26a1a77J4 volts bad	a26a1a67J7 gate_select bad	s	t
59 a26a1a77J4 volts bad	a26a1a68J7 gate_select bad	s	t
60 a26a1a77J4 volts bad	a26a1a69J7 gate_select bad	s	t
61 a26a1a58J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
62 a26a1a59J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
63 a26a1a60J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
64 a26a1a61J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
65 a26a1a62J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
66 a26a1a63J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
67 a26a1a64J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
68 a26a1a65J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
69 a26a1a66J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
70 a26a1a67J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
71 a26a1a68J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
72 a26a1a69J7 gate_select bad	a26a1a16 mplxr_gates faulted	s	t
73 a26a1a15-13 not_gate_1 bad	a26a1a20J2 gate_1 bad	s	t
74 a26a1a16 mplxr_gates faulted	a26a1a20J2 gate_1 bad	s	t
75 a26a1a15-15 not_gate_2 bad	a26a1a20J3 gate_2 bad	s	t
76 a26a1a16 mplxr_gates faulted	a26a1a20J3 gate_2 bad	s	t
77 a26a1a15-6 not_gate_3 bad	a26a1a20J4 gate_3 bad	s	t
78 a26a1a16 mplxr_gates faulted	a26a1a20J4 gate_3 bad	s	t
79 a26a1a15-7 not_gate_4 bad	a26a1a20J5 gate_4 bad	s	t
80 a26a1a16 mplxr_gates faulted	a26a1a20J5 gate_4 bad	s	t

Fig. 2 (cont) - Sample list of rules for module *A16*

pseudoterminal concept. For some modules within the sonar system, all of the input terminals may have effect at a majority of the output terminals. For modules with many terminals and that have parallel effect at the output terminals, the equation for the number of rules becomes:

$R = T_i \times T_o$

R is the number of rules resulting

T_i is the number of input terminals, and

T_o is the number of output terminals

If, however, pseudoterminals are used, the relationship of the number of rules to terminals is additive: $R = T_i + T_o$

The savings become evident when at least two input terminals exist and the sum of the number of input and output terminals is ≥5. Considerable savings in the number of rules can result from this strategy. Those savings translate into reduced loading and running time for FIS.

The format is the same as the one developed for the first strategy. However, the terminal name of either the cause or the effect will be a pseudoterminal, depending on whether an input or output related rule is being considered. While the pseudoterminal name can be any combination of characters, provided the first character is a letter, it is best to choose a descriptive terminal name as in Fig. 2, where *a26a1a16* is used. Also, a parameter and malfunction must accompany the pseudoterminal. This completes the cause or effect expression. The parameter used with the pseudoterminal can be any group of alphanumeric characters (provided an alphabetical character leads the group), but a descriptive, yet not real, parameter is preferred such as *mplxr_gates* as shown in Fig. 2. "Bad" is the preferred description of the malfunction state, although other descriptions such as faulted, in the example, are acceptable. For the sonar system, the convention used for naming the pseudoterminal was to abbreviate the module name. The parameter conventionally chosen is the word "function," and the malfunction condition was bad. The pseudoterminal cannot use the whole module name since there would be a conflict of a module name being defined as a terminal name, an unacceptable FIS syntax. In that case a syntax error results during compilation by using the FIS compiler. In the example, the terminal name is *a26a1a16*, which is an abbreviation of the entire module name *a26a1a16_mplx_gates*.

The third strategy implemented in the knowledge database formation is rules that express an abnormal functioning condition as the state of the normal function, i.e., a "false good." This strategy is used for modules that display status information. For the sonar system, the modules included are indicator lights, built-in-test (BIT), displays, and other lighted indicators. For these types of modules a bad input will not always result in the effect being bad, since there is ambiguity resulting from the effect of a faulted module or display. In either, a good or bad indication could occur independent of the actual fault state in the monitored area. Figure 3 contains an example of this situation, where an error light, *CORRELATOR TEST ERROR XDS8*, malfunction can express a "false good."

The result is that it becomes necessary to define a rule in which several causes will provide the same effect. This requirement prompted the development of "and" rules. "And" rules take their name from the connector & used to link the causes. "And" rule constructs have several causes complete in themselves, but separated by spaces and the connective &. A single effect follows the compound cause, and all other attributes of a rule remain the same. While this is an effective method for displaying the logic of the occurrence, it is not an accepted format for the expression of a FIS rule. FIS does not accept this format structure. To express this condition in actual implementation, a single rule or shallow sequence of rules replaces the compound rule. This directs the ultimate cause to a local module whose malfunction will produce a false indication. The compound condition requires a test to link the rule to a measurable state. The example displays such a replacement:

1 a26a1a27_driver *ADS8 logic_gate open*
2 ADS8 logic_gate open *ADS8 light on.*

In this example a shallow sequence (two levels) of rules describes the logic. The *a26a1a27_driver* for the indicator light *ADS8* is bad, which results in the logic gate being open. With the logic gate open, the light is on when it should be off. This situation requires care during rule description to allow consideration of all relevant causes. For comparison, the "and" rule to describe the relationship is:

1 a26a1a27_driver & ADS8 logic_gate open ADS8 light ok

where the & provides the logical connection of causal states resulting in a malfunction. The localized effect is unrelated to the signal fault. Also "ok," defined as a state free of malfunctions, is used as the state descriptor, thus a false good. This is a syntactically improper rule state for FIS. In the shallow sequence of rules, which replaces the compound rule, a syntactically correct expression "on" replaces "ok." "On" expresses the intent to the user, while at the same time expressing to FIS that a malfunction state is occurring. FIS only recognizes "ok" as a functional condition.

Cause - The cause has one of three formats in database: atom, triple, or compound (where the implementation replaces compound causes).

The atom is the simplest form, since it is just the name of the module to which the rule refers. In Fig. 2, rules 37 through 48 express this type of cause. The format for the name is the same as the format for the module name defined earlier. These must be exactly the same as the module name, or a syntax error will result when the FIS compiler is used. The atom form describes a faulty module producing any number of bad effects. The module is identified as faulty because the input is within specification, but the resulting output is out of specification.

The triple form is the most common type of cause used in FIS rules. It consists of three elements: a terminal, a parameter, and a condition. These three elements define a form used by FIS to identify the cause "list" in LISP. The terminal name defines either a physical terminal that exists at a spatial location, such as an edge connector, or it may be a logical creation, such as pseudoterminals or pseudonodes. The format used for the terminal name is the same as the module named for all physical terminals, except that a terminal designation is appended:

xxxxxxxJ16

as the *J16* does in this example. In the convention for the sonar system a *J* denotes a terminal designed as a test-point, as seen in rule 1 of Fig. 2. In other cases, a "-," seen in rule 13 of Fig. 2, indicates that the terminal could be an edge or pin connection that would not readily be testable. The relationship of these rules, in Fig. 2, relates to their physical counterpart in Fig. 1. Finally, a pseudonode or pseudoterminal has an abbreviated module name or module name with a suffix, which can have significance related to the type of rule. This type of terminal was exemplified by rule 74 in Fig. 2. Whether the suffix is added to the module name or the module name is abbreviated depends on the module name and rule being described.

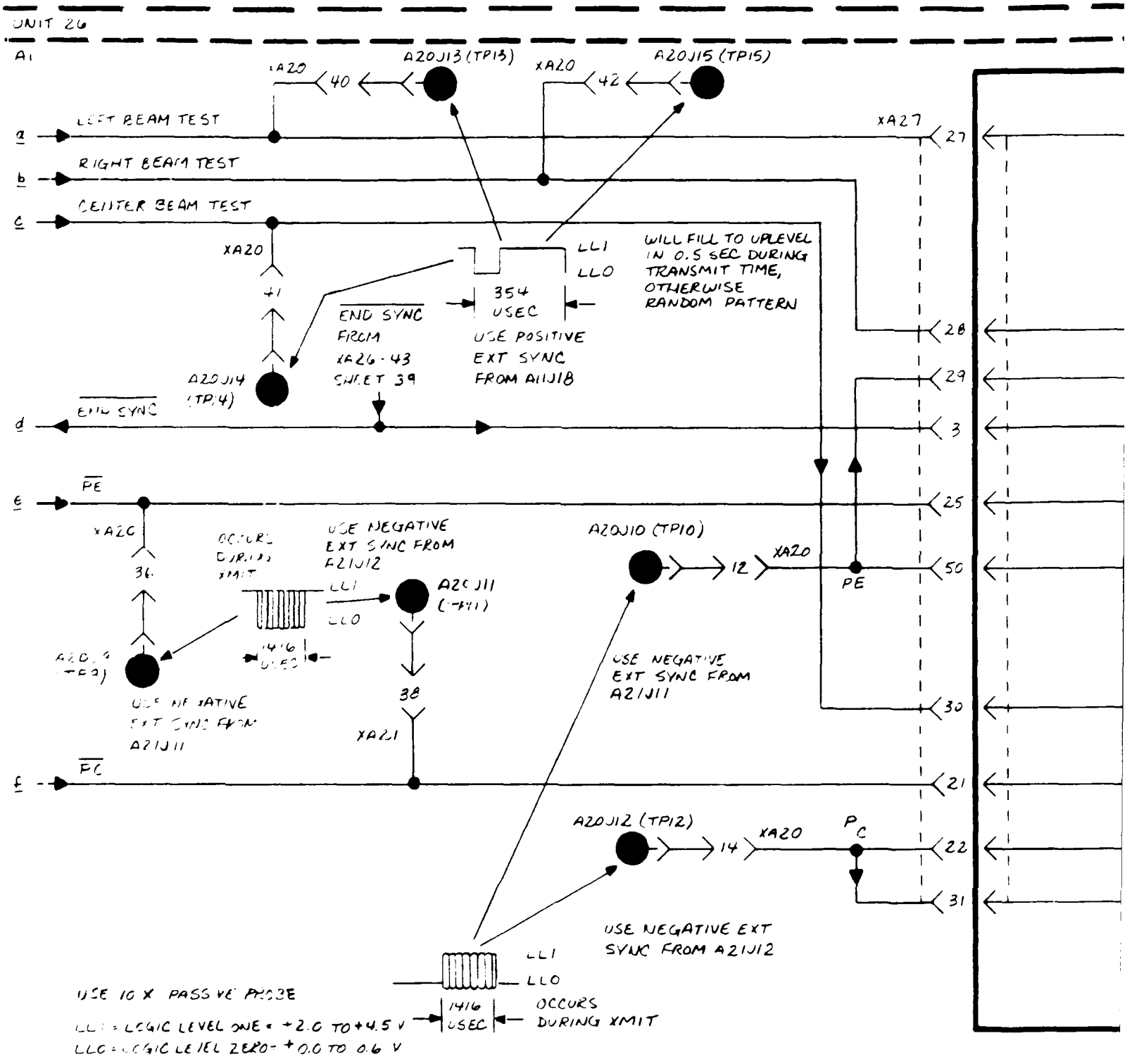
The cause parameter is either a measurable physical parameter or a logical creation that links the pseudo element in a set of rule constructs. Typical parameters relate physical quantities such as volts or frequency; others refer to system quantities such as a signal type, while others refer to more general quantities such as logic levels.

The parameter is a single group of characters separated from the terminal and state by spaces.

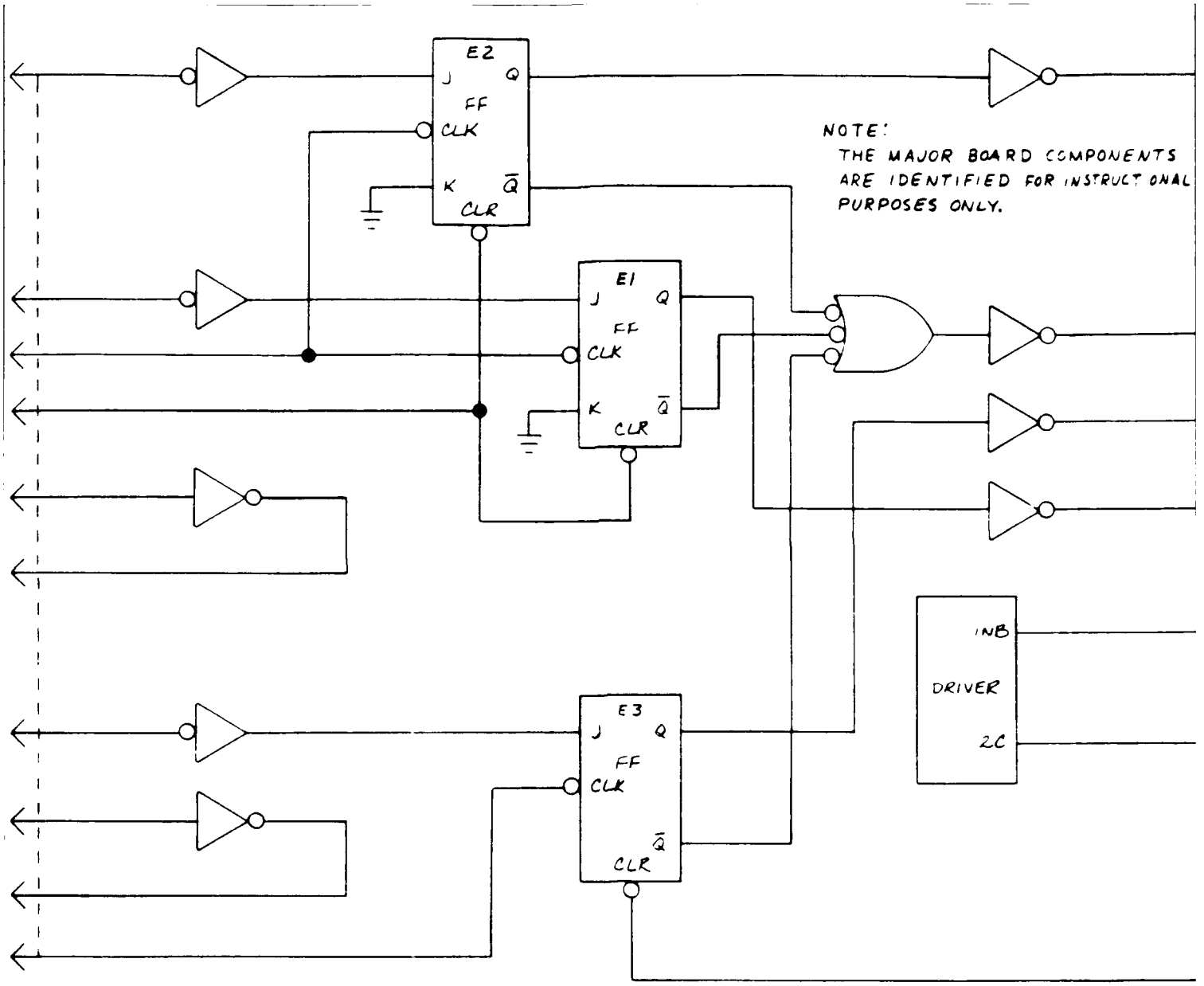
The state of the system parameter at a terminal can be in either fully functional or malfunctional. In analog systems, it is common for multiple degrees of a malfunction state to exist. Conditions to express the degree of each malfunction state must exist in the rules.

The state expression has one of two possibilities, either "ok" or any other state. The only acceptable reference to an unfaulted condition is "ok." Any condition other than "ok" identifies a fault. However, due to the syntax FIS requires, "ok" conditions are not a part of the rule set. The rules therefore only contain conditions for the malfunctioning state. The "ok" condition is relevant for the compound cause however. Common faulty conditions are: bad, high, or low. However, any other name for a faulty condition is permitted to express the degree of the malfunction state, such as: very-high, marginally-high, marginally-low, and very-low. The condition is a group of characters beginning with a letter of the alphabet, and it is separated from the parameter and the effect by spaces.

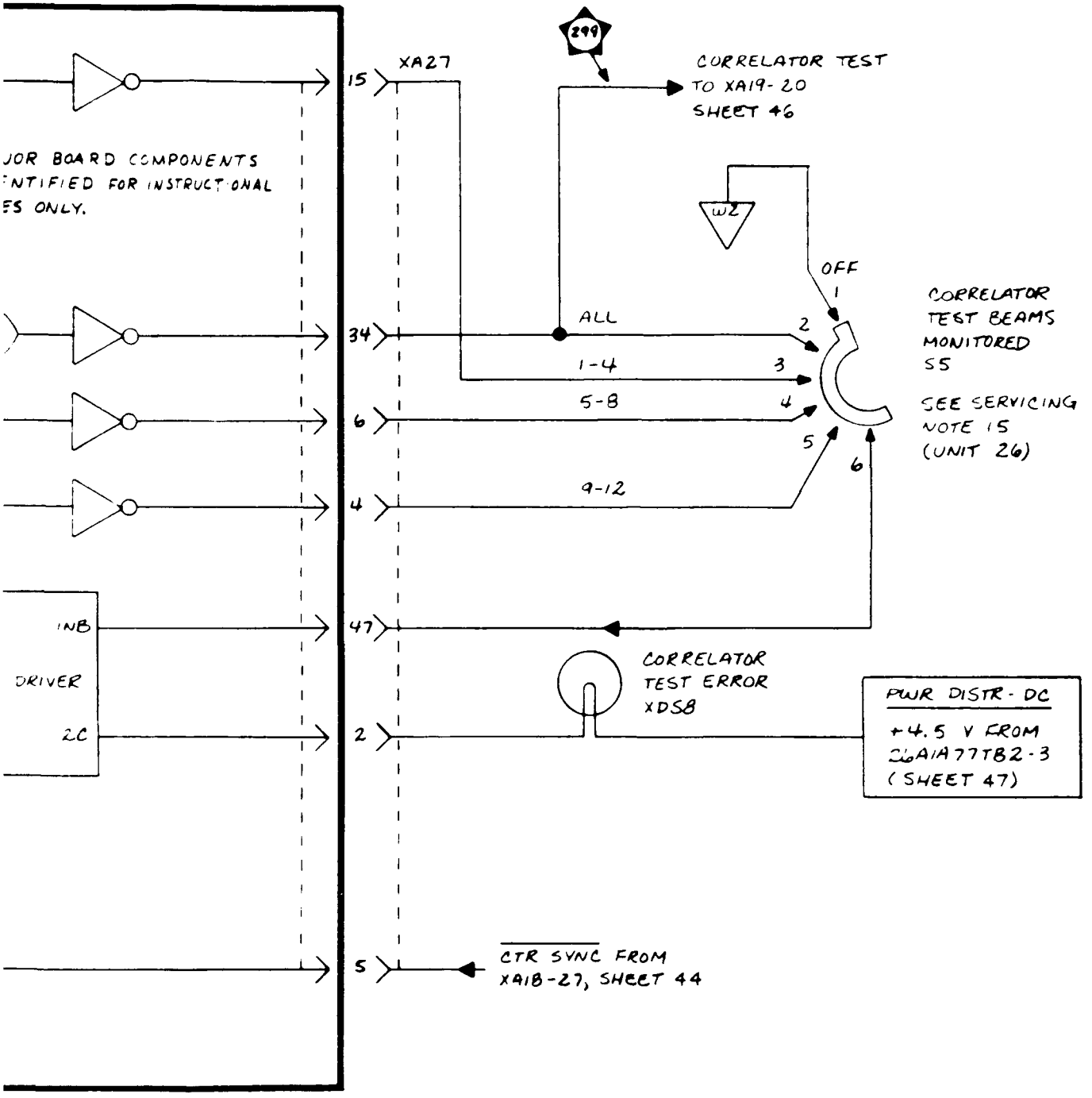
The compound form uses a group of causes separated by spaces and *&*. The individual causes may be either atom or triple type causes, or a mixture of both types. At present, the number of compound cause elements has no limit, but appears unlikely that more than one atom element would apply in a single compound rule. There could, however, be a very large number of triples present in a single compound rule. The compound cause with its associated effect and precondition must later be restructured to conform to syntax conventions. The restructuring entails replacing the compound rule with a series of simple rules to be compatible with the FIS format. The compound rules are useful in maintaining the thought process involved in the development, and to check the logic for the series of rules that replace the compound rules.


UNIT 26
A1
A20J13 (TP13)
A20J15 (TP15)
XA20
40
42
XA27
27
LEFT BEAM TEST
RIGHT BEAM TEST
CENTER BEAM TEST
a
b
c
d
e
f
XA20
+1
A20J14 (TP14)
END SYNC
LLI
LLO
354 USEC
END SYNC FROM XA26-43 SHEET 39
USE POSITIVE EXT SYNC FROM A1J18
WILL FILL TO UPLEVEL IN 0.5 SEC DURING TRANSMIT TIME, OTHERWISE RANDOM PATTERN
28
29
3
25
PE
XA20
36
OCCURS DURING XMIT
USE NEGATIVE EXT SYNC FROM A2J12
A20J11 (TP11)
1416 USEC
A20J9 (TP9)
USE NEGATIVE EXT SYNC FROM A2J11
38
XA21
A20J10 (TP10)
12
XA20
PE
50
30
FC
21
A20J12 (TP12)
14
XA20
PC
22
31
USE NEGATIVE EXT SYNC FROM A2J11
USE NEGATIVE EXT SYNC FROM A2J12
USE 10 X PASSIVE PROBE
LLI = LOGIC LEVEL ONE = +2.0 TO +4.5 V
LLO = LOGIC LEVEL ZERO = +0.0 TO 0.6 V
1416 USEC
OCCURS DURING XMIT

GTA41614A



TEST MONITOR B (3TG10) A27



; that would result in a "false

Effect - Only one format is allowed for the effect. It is the same format as the triple format for the cause. All three elements - the terminal, parameter, and state are indistinguishable from the cause structure. This is reasonable, since for several modules connected together the effect in one module becomes the cause for the next module. The effect and cause relationship provides a link to express the relationship of the physical connections between system modules. This linking effect can be seen in the example of the pseudoterminal in Fig. 2. The triple relating the effect of rules 61 through 72 becomes the cause of rules 74, 76, 78 and 80. This linking relationship becomes more evident when examining the rules for several modules that are physically linked.

Type — The type is presently just a place holder in the current implementation of the FIS database. In previous implementations two possible types existed; sometimes *s*, and always *a*. "Sometimes" type rules indicate a degree of fuzziness for a rule, so that the bond between the cause and effect are weakened. "Always" type rules indicate that a specific cause would be certain to produce a given effect. Because of the probabilistic nature of FIS, it was later determined that only "sometimes" rules would be viable, since "always" rules would produce direct implications on module viability. Therefore, the present knowledge database can have either *s* or *a* as the type. During the conversion to a LISP compatible format only type *s* is a valid default state for all rules. Later implementations of the knowledge database will not contain the type designation.

Precondition — The precondition is a single descriptive atom that describes the condition the UUT should be in while the test is being made. The default value is *t*, which occupies the remaining space in the FIS database format. This form can be seen in Fig. 2. The default state is generally the most common condition of the system during testing. Other than the default, any group of characters can act as a precondition, provided the first character is a letter. Figure 4, which is an excerpt from the rules for the *s9* module, displays an example of a precondition other than the default. Since *t* is the assumed default, later data structures may choose to eliminate the precondition item and default to *t*, unless a unique precondition exists.

Module: a26a1S9

No	Cause	Effect	Type	Precondition
1	a26a1S9-1 time_slot_1_volts hi	a26a1a24J1 time_slot_1_volts hi	s	a26a1S9_set_to_oper
2	a26a1S9-1 time_slot_1_volts lo	a26a1a24J1 time_slot_1_volts lo	s	a26a1S9_set_to_oper

Fig. 4 - Excerpt from rule list for module *a26a1S9* exemplifying a nondefault precondition

Optional Items

All text items that follow either a rule number or the word "module" are considered to be information necessary to FIS. If other information that is only for human consumption is present in these locations then that information is considered optional and may only be included if delimited on the left by a */* or *{*, and on the right by a */* or *}*, respective of matching the first delimiter. A comment closing delimiter must occur on the same line as the beginning comment delimiter to identify the item as a comment.

Other optional items desired in the file may occur without delimiters provided they are not in a module or rule location. Several types of this information are found to be useful in the rule database for the sonar system.

For the purposes of data management the following items provided valuable optional information:

a. A data title to describe the file information;
b. The date of the rule set version;
c. Column headers, such as No (Number), Cause, Effect, Type, and Precondition increase the readability;
d. If a rule is deleted, replacement of the rule with a comment, such as [deleted, date];
e. Page numbers;
f. Any other comment about the state of the file.

Each of these features are found in Fig. 5, except the page number that would be in a standard location such as at the bottom center of each page.

Left Correlator Rule Base
July 1989

Module: a26a1a58_mod_amp_mpx

No	Cause	Effect	Type	Precondition
1	a26J3_beams volts hi	a26a1a58J1 volts hi	s	t
2	a26J3_beams volts lo	a26a1a58J1 volts lo	s	t
3	a26J3_beams uniformity bad	a26a1a58J1 volts hi	s	t
4	a26J3_beams uniformity bad	a26a1a58J1 volts lo	s	t
5	a26J3_beams waveform bad	a26a1a58J1 waveform bad	s	t
6	a26a1a58J1 volts hi	a26a1a58J3 volts hi	s	t
7	a26a1a58J1 volts lo	a26a1a58J3 volts lo	s	t
8	a26a1a58J1 waveform bad	a26a1a58J3 waveform bad	s	t
9	a26a1a70J3 reference_signal bad	a26a1a58J3 volts hi	s	t
10	a26a1a70J3 reference_signal bad	a26a1a58J3 volts lo	s	t
11	a26a1a70J3 reference_signal bad	a26a1a58J3 waveform bad	s	t
12	[Deleted 21 July 1987. Moved to a26a1FL1.]			
13	[Deleted 21 July 1987. Moved to a26a1FL1.]			
14	[Deleted 21 July 1987. Moved to a26a1FL1.]			
15	[Deleted 21 July 1987. Moved to a26a1FL1.]			
16	[Deleted 21 July 1987. Moved to a26a1FL1.]			
17	[Deleted 21 July 1987. Moved to a26a1FL1.]			
18	a26a1a58J4 volts hi	a26a1S9-1 time_slot_1_volts hi	s	t
19	a26a1a58J4 volts lo	a26a1S9-1 time_slot_1_volts lo	s	t
20	a26a1a58J4 waveform bad	a26a1S9-1 time_slot_1_waveform bad	s	t
21	a26a1a58_mod_amp_mpx	a26a1S9-1 time_slot_1_volts hi	s	t
22	a26a1a58_mod_amp_mpx	a26a1S9-1 time_slot_1_volts lo	s	t
23	a26a1a58_mod_amp_mpx	a26a1S9-1 time_slot_1_waveform bad	s	t
24	a26a1a58J7 gate_select bad	a26a1S9-1 time_slot_1_volts hi	s	t
25	a26a1a58J7 gate_select bad	a26a1S9-1 time_slot_1_volts lo	s	t
26	a26a1a58J7 gate_select bad	a26a1S9-1 time_slot_1_waveform bad	s	t
27	a26a76J3 volts bad [+12v supply]	a26a1S9-1 time_slot_1_volts hi	s	t
28	a26a76J3 volts bad	a26a1S9-1 time_slot_1_volts lo	s	t
29	a26a76J3 volts bad	a26a1S9-1 time_slot_1_waveform bad	s	t

[Modified 21 July 1987. a26a1a58_mod_amp_mpx: Moved rules 12-17 to a26a1FL1.]

Fig. 5 - Excerpt from rule list for *a26a1a58* module displaying the format for comments

Testlist File

While the causal rule file defines the interrelation of all of the modules, it does not contain useful information for the technician or automated test equipment. The testlist file contains all the basic information necessary to perform a test, except explicit instructions. FIS uses the information contained in the testlist file to suggest to the user which tests will provide the greatest amount of information about the system's state; the best test. With the rule information, FIS correlates the information so that each real test provides implications on the health of modules. The testlist, when converted to an FIS format, then provides the information necessary for the inference engine to calculate the effect of performing a test.

For the sonar system, the information contained in the testlist was obtained from the actual tests in the troubleshooting process, as given by the sonar system manuals [2,3]. The tests defined are based on the use of real test points designed into a system and that are readily accessible. In addition to measurable test points, tests that include reading indicator lights, displays, or monitors are visual tests, and their interpretations provide the test result. Figure 2, from Ref. 2, displays an example of general test information that is available from the module level schematics. If additional information to complete the test definition is needed, the knowledge engineer should refer to a detailed schematic such as is found in Fig. 6 [3]. Figure 6 provides a descriptive procedure in addition to detailed schematics. This information should be sufficient to describe tests in the correct information format and will give sufficient detail in creating the instruction information for FIS.

Again, as in the causal rule file, two types of information exist. One type is information that becomes functional for FIS. The other type is optional information that is used in the management of the data. The following description of the required information is listed in the order in which the information should appear as columns across the page. The appropriate column header defines the column of information. The section on "optional information" presents a description of the header information.

Figure 7 shows an example of the format used for the sonar system test information with a more extensive example appearing in Appendix B.

Name

The name should be descriptive of the test. This item is created by the knowledge engineer and should be tailored to his needs. It must be an atom that has a letter as the first character. The convention used for the sonar system was to abbreviate the name of the test point, or to abbreviate the test point and add a suffix for cases that require additional identification. Such a convention allows the user, who will not be able to remember all of the test names, to more easily make tests in FIS by using the make test name (mtn) command. If the user wishes to by-pass the best test information, an intuitive naming structure simplifies the function of making a test. If, however, the name is not intuitive, the make test (mt) command still allows the user to enter the terminal. FIS then prompts the user for the parameter and setup.

Other strategies have created names that are just test numbers led by the letter "t." These strategies have primarily been conceived for Test Program set generation or use with automatic test equipment. If this strategy is used for technician aid applications, a method must be developed for the user to identify the name.

Test point

The test point should follow the same format as the terminal used in the causal rule file. A correlation must exist between the terminal name used in the causal rules and the terminal identified in the test information file; they must be exactly the same names.

If the test points do not correspond, FIS will signal that syntax errors exist. If the knowledge engineer ignores the syntax errors, FIS will compile the knowledge database, but it will lack of logical connection between tests and rules. Errors will result when using the knowledge database. Therefore, it is important that the knowledge database be cross-checked for such occurrences of syntax errors.

Parameter

The parameter also should relate to the causal rule file and have the same format. The purpose is to name the physical parameter that the user would measure in the performance of a test. Again, if the parameter does not match the parameter as expressed in the rule, then the elements of the knowledge database will not maintain their continuity.

Units

The unit item is similar to the parameter, but it defines precisely the parameter unit to measure. For example, the units for the parameter voltage (or volts) could be volts, millivolts, microvolts, etc. The "unit" indicates that a quantitative value has a unit that must be consistent with the test procedure. Primarily, the format requires units so that the technician when using FIS will enter an appropriate quantitative value to provide an accurate test result.

Qualitative Values

At least two qualitative values are presumed, one naming the functional state and all others defining a malfunction. The only qualitative functional value acceptable to FIS is "ok." All other qualitative values represent malfunctioning states. The qualitative values would also correlate with the conditions described for the causal rule file. Thus, for example, if five different types of malfunction exist, then there should exist five types of rules with the appropriate states identified. The malfunction states arise from the levels of performance variation.

The format for the qualitative values is that they must occur in a column with no line spaces between the entries in the column. It is not necessary that "ok" always be at the head of the column, but it is a good practice.

Minimum Quantitative Value

This number defines the minimum value that is acceptable as a functional "ok" reading. By definition, any value less than the minimum quantitative value corresponds to a malfunction and relates to the relevant qualitative value. The range then defined extends from negative infinity to the minimum quantitative value; this will always be the case. If various levels of malfunction exist then the minimum will define the absolute lowest limit functionality, and the other states will fall outside the defined bound extending to negative infinity. Relating the value in Fig. 7 to the actual test point in module A70 of Fig. 1, the minimum value can easily be seen to correspond.

Maximum Quantitative Value

This number defines the maximum value that is acceptable as a functional "ok" reading. By definition any value greater than the maximum quantitative value corresponds to a malfunction and relates to the relevant qualitative value. The range defined extends from the maximum quantitative value to positive infinity. If various levels or malfunctions exist, then the maximum value will define the upper limit of functionality, and other states will fall beyond the defined bound to extend to positive infinity. Similarly, the relationship of the physical measurement of Fig. 1 can be correlated with the value of the maximum value given in Fig. 7.

Cost

The cost is a value defined in terms that assume an approximate cost of the actual performance of the test. In an earlier version, FIS uses cost as one of the criteria for selecting a test as a "best test." The cost item is not used in the present version. If implemented, the cost value is generally a simple number, but it may be an equation for computing the value.

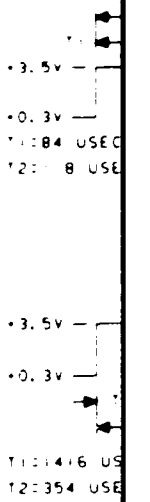
CIRCUIT DESCRIPTION

The network coupler accepts logic levels and provides output alarms when any of the inputs are absent. The network coupler consists of six circuits. Input levels for the circuits are 2.0 to 4.0 vdc (high level) and 0 to 0.6 vdc (low level). Circuits 1, 2, and 5 contain a mixer network with a long time constant that determines the duty cycle. The output voltage levels are a function of the duty cycle of the corresponding input signal. The normal inputs must be continuously present for the outputs to remain less than ±150 mv. In circuit 1, logic signals are applied through finger contact 4 and diodes CR1 through CR3 to the base of inverter Q1. Q1 collector output is applied through a long time constant network (C1, and R1 through R6) to finger contact 29. The output at finger contact 29 is a function of the average dc voltage at Q1 collector and the input duty cycle. The output increases at finger contact 29 above the alarm level of +150 mv when the input duty cycle drops below approximately 17 percent (20-usec positive mv width out of a 118-usec period), and decreases below the alarm level of -150 mv when the input duty cycle raises above approximately 85 percent (140-usec positive pulse width out of a 1416-usec period). Any steady state input of a logic low level or high level causes an alarm voltage which exceeds ±150 mv.

Except for duty cycles, circuit 2 functions are similar to circuit 1. The logic levels applied to finger contact 5 operate the circuit of inverter Q2 in a similar manner to that of Q1.

The output increases at finger contact 30 above the alarm level of +150 mv when the input duty cycle (1416 usec) drops below approximately 10 percent (140-usec positive width), and decreases below the alarm level of -150 mv when the input duty cycle rises above approximately 35 percent (500-usec positive width). Any steady state input of a logic high or a logic low causes an alarm voltage that exceeds ±150 mv.

Circuit 3 combines 12 input multiplexer gates at a 12-input NOR gate (CR7 through CR32) and applies them to the base of inverter Q3. The inverted composite of the inputs appear at Q3 collector and J3. Under normal conditions, the inverted composite waveform is a continuous uniform logic signal. The inverted composite waveform is coupled to U1, through diode CR33, to integrator R28 and C3 to the base of emitter follower Q4. The inverted composite waveform from U1 is coupled through CR34 to integrator R30, C4, and C5 and applied to the base of Q5. With normal inputs, both integrator voltages remain below +3 volts and emitter followers (used as OR gates) Q4 and Q5 are cut off. When a fixed low level is applied, the collector of Q3 becomes more positive, C3 charges to a level greater than +3 volts, Q4 conducts more heavily, Q6 cuts off and the emitter of Q7 goes down to near ground potential. This sets U2 to the alarm state and output mixer R37 through R39 at finger contact 42 drops to below -150 mv. When a continuous high level is applied, the collector of Q3 goes less positive and the output of U1-2 goes more positive. As a result, capacitors C4 and C5 charge to a level greater than +3 volts, Q5 conducts more heavily, and Q6 cuts off. With Q6 cut off, emitter follower Q7 co ducts, U2 is set to the alarm state and the voltage at fi ger contact 4 drops to below -150 mv.



The fault locator transmit pulse at finger contact occurs at a slow rate compared with the multiplexer sign rates. The pulse (a logic high level) is coupled through t inverters in U1 to sharpen the waveshape. The output of U1 is directly coupled to U2-10 which is the clear input a resets the flip-flop periodically, so that the alarm lev will not remain when the fault condition is only temporary

In circuit 4, a logic level is applied to finger contact 18, through diode CR35 and voltage divider R41,R to finger contact 43.

In circuit 5, a 4-MHz square wave is applied to finger contact 19. The output at finger contact 44 is a nominal volt if the input is 0.3 to 3.5 vdc. When the input is continuous high or low level, the output of mixer netwo (R43 through R46, and C6) changes to a ± 500 mv ala output.

Circuit 6 consists of an input at finger contact which is applied to inverter U1. The inverted signal outp from U1 is applied to finger contact 45.

NOTE

This unit may have part number 77D602227G1 or 77D611979G1. Both units are electrically and physically identical, although component differences do exist. If repair of the unit is required, the proper replacement component can be selected from the parts list given in DATOM Manual 8 SE313-TP-MMC-110.

off, emitter follower Q7 con-
n state and the voltage at fin-
-150 mv.

nit pulse at finger contact 17
ed with the multiplexer signal
h level) is coupled through two
e waveshape. The output of U1-3
which is the clear input and
cally, so that the alarm level
t condition is only temporary.
level is applied to finger
35 and voltage divider R41,R42

luare wave is applied to finger
nger contact 44 is a nominal 0
3.5 vdc. When the input is a
, the output of mixer network
changes to a ± 500 mv alarm

an input at finger contact 20
U1. The inverted signal output
contact 45.

DTE

t number 77D602227G1 or
s are electrically and
though component differ-
of the unit is required,
component can be selected
iven in DATOM Manual 8

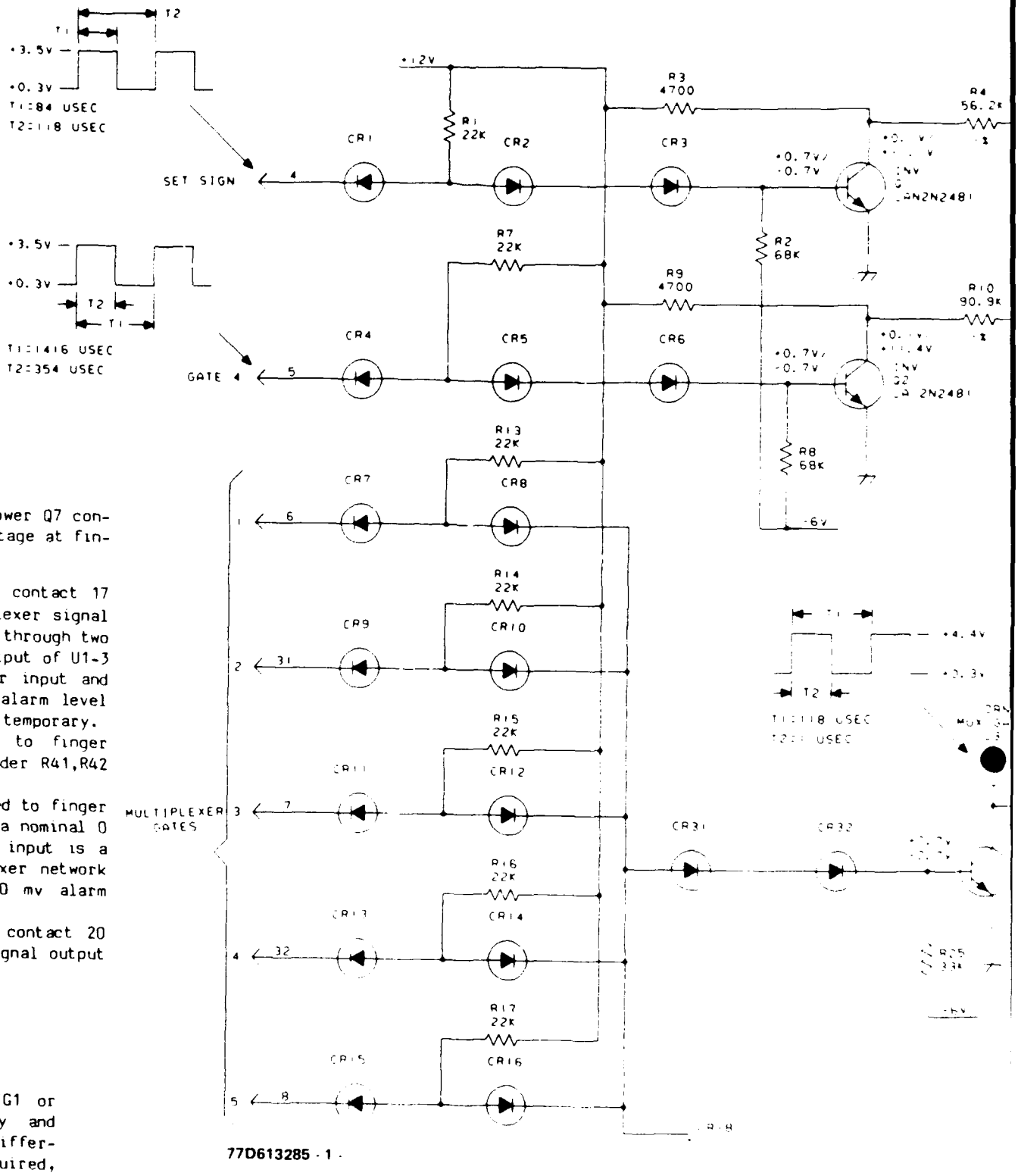
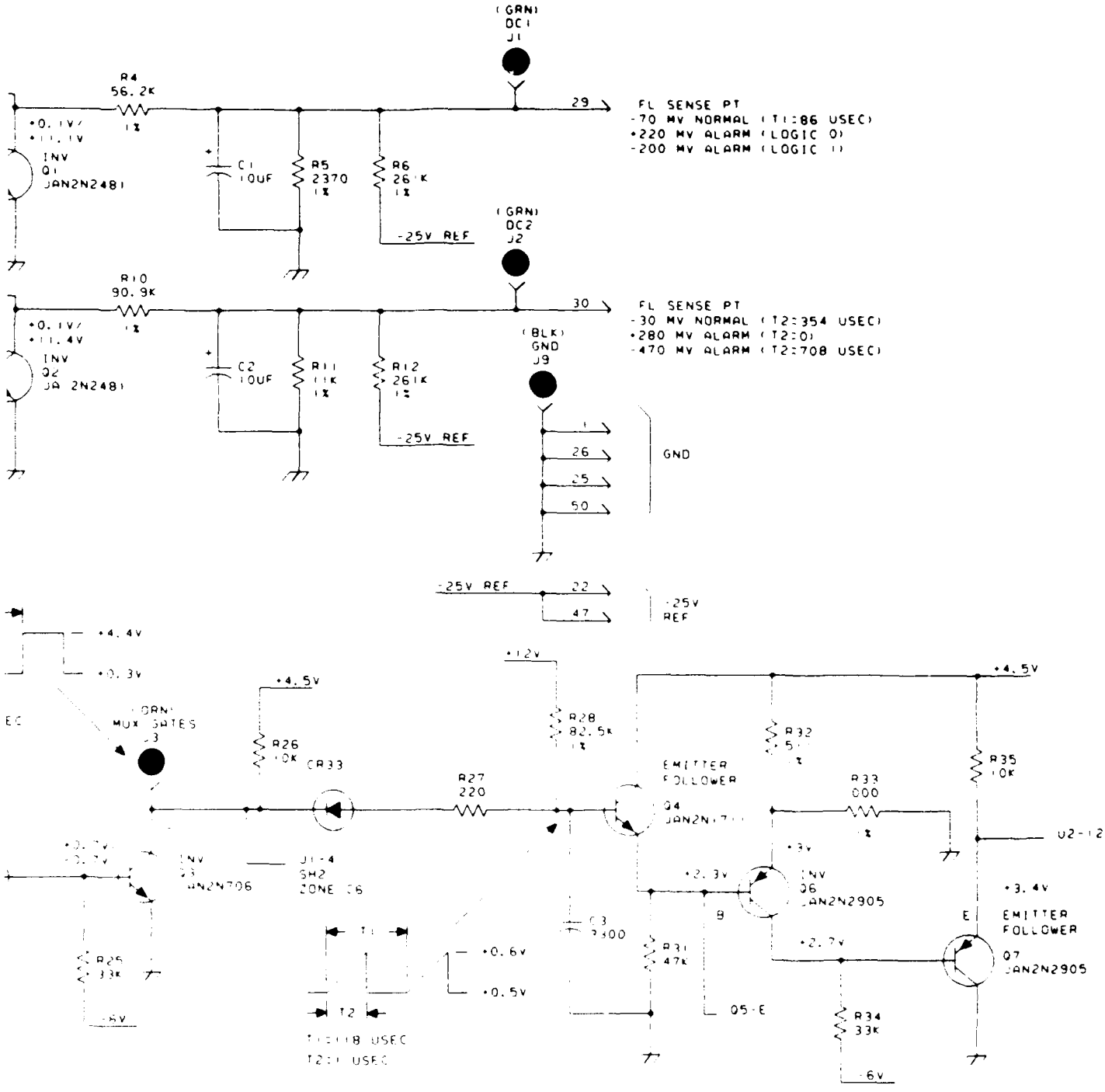


Fig. 6 - Detailed schematic diagram for Unit 26 of the AN/SQS-53B, which displays component level informati and detailed test procedure information (primary focus is placed on module *26A1A19*) [3]



SCHEMATIC DIAGRAM

Diagram is provided for information only. The assembly is not repairable onboard ship.

onent level information
1A19) [3]

NAME	$1 TEST POINT	$2 PARAMETER	UNITS	$3 POSS QUAL RES	$4 MIN	$5 MAX	COST, SECONDS	PREREQUISITES	INSTR NAME	TEXT PARAMETERS
A70J7_scope	a26a1A70J7	volts	volts	hi lo ok	6.8	8.4	20	unit-26_door open a26a1_drawer open diff_scope ready probe on a26a1A70J7	v_dif_scope	$6=26A170J3 $7=peak to peak

Fig. 7 - Excerpt from testlist, displaying the format of optional and required information

Prerequisites

This column of values defines the default state the system must be in for the test cost to be of the value stated in the cost column. The format is that a prerequisite must have two sets of characters separated by a space. In general, the purpose of the prerequisites is to display the state of the system or test instrument. FIS does not use this item; the sole purpose is to define "like tests" for the knowledge engineer to use in managing the data.

Instruction Name

FIS uses items in this section and the next section to display operational instructions to the technician. These items provide the means of correlating the tests that contain specific information with the general instructions.

The "Instruction Name" is a single set of characters beginning with a letter and relating to a test instruction procedure that occurs in the instruction file. The instruction name identifies with several tests because the instruction provides general information for a class of tests. By relating the test to the instruction several advantages can be realized, such as reduced disk usage, reduced RAM requirement, and reduced loading time for the knowledge database. Also, in the knowledge database format the knowledge engineer can readily relate tests to instructions.

Text Parameter

The text parameters also relate to the instruction file and provide specific test instruction information to the instruction command of FIS to tailor the generic instruction to a specific test. The purpose is to reduce the number of instructions needed in the file. The text parameter can be of any format. A *$* and numbers identify the field, which relates to a *$* and number in the instruction. During operation the fields, identified as text, fill into the instruction blanks for each test.

Column Headings

Column headings occur above the columns containing the data. The actual column heading can have any name, but at least one space must separate the headings. Overlapping column headings are not allowed. The complete heading consists of four lines. As Fig. 7 shows, the first line may only include the characters *$3* (where *$3* indicates a column field that provides information to the instruction file) and spaces. The format uses *$3* to identify that the following three lines are lines of the header and, secondly, to mark the qualitative value column. This is just one of the columns marked with a *$* followed by a number. These columns contain information that passes to the instruction file when processing an instruction for a test. While the numbers relating to the column information are not rigid they must correspond to the information required for a corresponding *$* numbered space in the instruction file. Therefore, the format used for the sonar system has the following *$* numbers corresponding to the column information stated:

$1 - Test point
$2 - Parameter
$3 - Qualitative value
$4 - Minimum quantitative value
$5 - Maximum quantitative value
$6, $7, $8, $9 and $0 - Contained in the text parameter column.

Setup

The setup consists of a single field of characters used primarily to differentiate tests made at a common test point. It can also identify a unique testing condition that would be common for several tests. This item was not in the original format as seen in Fig. 7. It was added through the FIS editor before compilation. The format was later revised to include a column for the setup.

Test Type

The two types of tests may be either performance or diagnostic. Performance tests determine if the system is working, if it is out of specification or has some parameters misaligned. Performance tests "P" isolate to the functional unit where the fault is occurring. Diagnostic tests "D" are those tests used once a fault has been determined to exist. This occurs only after a performance test has failed. Diagnostic tests isolate to the faulty module. This item was not in the original format as seen in Fig. 7. It was added through the FIS editor before compilation. The format was later revised to include a column for the test type.

FIS uses the test type to prioritize the best tests. FIS first selects performance tests as the best test. This continues until the performance tests are exhausted, or a performance test has provided a result out of specification. Upon completing a performance test that indicates a malfunction, diagnostic tests become available for FIS to suggest as best tests. If a "bad" performance test occurs, then FIS has the ability to select the appropriate best test from the combined set of all performance and diagnostic tests. In this case, the heuristic search criterion determines which best test FIS will provide.

Optional Items

An optional item may appear anywhere in the file. They are delimited on the left by */* and on the right by */*, or they must have an * as the first character. Anything appearing in the file that is not part of a column heading is data. Thus, page numbers, dates, titles, page headers, comments, etc. must appear as optional items.

Remaining Notes on the Testlist File Structure

Data for a particular test do not have to appear on consecutive lines, i.e., comments and blank lines may break up a block of data. This is useful, for example, if information of a column extends onto the next page. However, a new column header may not occur to break up the block of data.

Precondition File

The precondition file links the preconditions established in the causal rule file with specific tests. The specific precondition will apply when preparing to perform a test, since it implies that a particular condition exists to allow the rule to fire. The precondition file also defines if any particular sequence is required to perform a test. Because of the simplicity of this file, direct preparation in the LISP format becomes possible, and eliminates the need for conversion from a human-readable format. However, should direct preparation of this file become inconvenient, a conversion could be developed. The two essential elements of the precondition file are the file title and the precondition definitions. The precondition file is created by the knowledge engineer from the available information used to create the rule preconditions. In general, these arise out of unique switch states that must exist in the system to allow a different signal to pass or a connection to exist. An example of a precondition file is supplied in Appendix C.

The File Title

The name of the file should coincide with the name for the causal rule and the testlist files, and it must have the *.prec* suffix instead of the respective *.rule* and *.test* suffixes. This title appears only in the naming of the file and not in the file itself.

Precondition Definitions

The preconditions that appear in the causal rule file apply to a set of tests identified by the test name. Also, since the precondition is a switch state, then the reverse switch state definition becomes a default by producing the negative precondition, or not precondition. The precondition should be a lisp atom, or, if necessary, more than one lisp atom. The following indicates the contents of the structure:

((Precondition_A (TestName_1 TestName_2 TestName_3))
(Precondition_A_Off (Not Precondition_A))
(Precondition_B (TestName_3 TestName_4 TestName_5))
(Precondition_C (TestName_6 TestName_7))).

If the precondition definition is unnecessary because of the nature of the system being analyzed, then it is appropriate to define the information set as *NIL*.

Order Information

The order information identifies tests that must be performed before other tests can be made. An example of such a test is when the system must be in a different functional state for two separate tests. Subsequent tests branch out of each of the tests performed, while the system is in different functional states. Thus, if the first test is omitted, the system is not in the correct state to perform subsequent tests. Two elements are provided; the file title and the order definition. An example of the structure and format is supplied in Appendix D.

The File Title

The name of the file should coincide with the name for the causal rule, testlist, and precondition files, but should have the *.ordr* suffix.

Order Definitions

The order required for testing defines branches - how certain tests must follow other tests in order of execution. The following indicates the contents of the structure:

((TestName_A (TestName_B TestName_C))
(TestName_D (TestName_E TestName_F TestName_G)))

In this example, two lists of orders exist for performing tests. The first is *(TestName_A (TestName_B TestName_C))*. Within this list, *TestName_A* is the name of a test that must be selected and performed before FIS allows *TestName_B* and *TestName_C* to become available to the user. Thus the order of performing *TestName_A* before the other two is important because *TestName_A* produces a state that affects the performance of the other tests. The requirements for ordering tests did not appear in the sonar system application. It could, however, have been implemented to require all of the tests with the same precondition to be performed in some order at the same time. In essence, the order list allows the knowledge engineer to have a means of defining a test sequence that should be performed by the technician. Similarly, in the example, *TestName_D* must be performed before *TestName_E*, *TestName_F*, and *TestName_G* become available tests. Whole sequences of tests can be defined by nesting the following test in parentheses after the test which should precede it.

If the order information is unnecessary because of the nature of the system being analyzed, then it is appropriate to define the information set as *NIL*.

Instruction File

The instruction file provides textual instructions to the technician during a test sequence. Three primary elements are in this file: the instruction name, instruction text, and string inputs acquired from the testlist file. Figure 8 shows an example of an instruction file. A more extensive excerpt is available in Appendix E.

```
Instruction Name: A45J5_a

Instruction Text:

"Connect oscilloscope to ",$1,".

Observe modulated pulse with duration of approximately 50 msec.
Measure the peak to peak amplitude of the pulse.
The correct range is,"$4", to, "$5"."
```

Fig. 8 - Excerpt of an instruction from that instruction file

Instruction Name

This item must appear as the words *Instruction Name* followed by a colon (:). The actual name of the instruction (that should be correlated with an instruction name in the testlist file) follows after any number of spaces, as shown in Fig. 8.

Instruction Text

This item must appear as the word *Instruction Text* followed by a colon (:). The actual text must follow after at least one space and a right quotation mark ("). Items following the quotation mark are part of the text until encountering the second quotation mark ("). Subsequent text enclosed in quotation marks may follow until the instruction text is complete. A break in quoted material usually is where string information from the testlist file occurs. In Fig. 8, all the generic text for several tests is enclosed by quotation marks, and the specific information for each test is supplied in the unquoted area marked by the string number.

String Input

The string input is a number, 1,2,4,5,6,7,8,9 and 0, led by the dollar sign *$*, as in the example, *$1*. The testlist information provides FIS with information to complete the instruction, permitting instructions to appear as complete text, with the addition of the linking test information.

During conversion to the LISP format, comments and any other optional information entered in the file are ignored. Only information in quotations or string variables identified by *$*, or the instruction name identifier *instruction name* are converted.

Conversion of Knowledge Database from Human Data Management Format to LISP Compiled Database

The information described in the previous chapter on knowledge database formation provides the substance that the FIS shell needs to act as a technician's assister. It is not, however, in a format that the FIS can use, except for the unaltered precondition and order files. FIS is written in LISP, therefore, it is necessary to translate the information from a format that is understood by humans to the language that the FIS can understand.

To perform this conversion, several programs were written in the C language. The computer used was a VAX 11/780 with a VAX/VMS 4.6 operating system. A version with slight modifications for UNIX was also created. It was convenient to select this machine for data management. Since FIS is implemented in a UNIX operating system environment, the files, once converted, were transferred to the computer running FIS. The conversion programs in Appendices F, G, H, I, J and K were portable to other computer and operating systems. Appendix F provides the automated procedure for rule and test conversions. Appendix G provides the semiautomated procedure for rule conversion. Appendix H provides the automated procedure for testlist conversion. Appendix I provides the semiautomated procedure for testlist conversion. Appendix J provides the procedure for extracting test instruction information from the testlist file. Appendix K provides the procedure for formation of instruction information. This Section provides the information on how to use these procedures.

The four conversion programs used provide a LISP consumable form for the rulelist, testlist, and instructions. The reason for the four conversion programs is that the testlist contains information that is required by both the UUT file and LISP instruction function of FIS. Therefore, two of the conversions provide the information, the testlist and instructions, to two different respective parts of the FIS palatable knowledge database. Once converted, the user appends the resulting files into a single file. This file is compiled to the executable FIS Knowledge Base.

Rule Conversion

Three types of rule conversion exist. One generates the rules into a FIS format, readable through the FIS editor. This type of procedure adds new modules and corresponding rules to an existing knowledge database, and is semiautomated. Another type of procedure is automated and results directly in the correct FIS format, with default values given for certain data items. This was the primary method used in the sonar system application. The third type of conversion enters the data manually by using the FIS editor. This last method was used to make minor changes.

In each method of conversion the rule database stores the essential information in a file accessible by FIS. All conversions eventually result in the same information existing in the knowledge database. The first type, however, demands a second step that requires human interaction with the FIS editor to arrive at a final format. The second type of conversion produces a final format, but some information, such as failure rates for modules, is given a default value that can be modified to an actual value. The third type is directly formulated as the user enters the information through the editor.

Semiautomated Rule Addition

In the first type of conversion, the module name, defined in the rule database, is stripped during the conversion process and an output file is generated with its name. The program contained in Appendix G is used to perform the conversion. When the program executes it uses the rule data file as the input file. Each of the module names, in the rule data file, is given to an output file opened with the same name. The output file generated contains all of the rules associated with that module.

The conversion generates a separate file for each module within the UUT, with the file name being the module name. The format of the output file is the precondition (defaulted at *t* in the example), the cause, and the effect

associated with each rule. The following is an example of the contents of the file generated from this conversion:

t power (a26_cabinet_pwr volts bad)
t (a26_cabinet_pwr volts bad)(a26a1a74j1 volts bad)
t (a26_cabinet_pwr volts bad) (a26a1a74j3 volts bad).

where *t* represents the precondition for each of the three rules. In the first rule, *power* represents the cause. In the second rule, the list *(a26-cabinet-pwr volts bad)* represents the cause. In the third rule, the cause of the second rule is repeated. The effect of the first rule is the list *(a26-cabinet-pwr volts bad).* The effect of the second rule is *(a26a1a74j1 volts bad).* And the effect of the third rule is *(a26a1a74j3 volts bad).* The cause may be either a module name or a triple in the form of a list, enclosed in parentheses, that consists of a test point, a parameter, and an abnormality. The effect is always a triple.

The conversion program prompts the user for the name of the file to convert. After entering this information, the conversion will occur resulting in new files, one for each module in the rule database. Thus, after the conversion, the working directory will contain as many new files as there were modules in the rule base. When entering a *ls* command in UNIX, the directory listing will include all of the files existing before the conversion and the new files generated in the conversion. If the converted file is not in a proper format, as described in the previous section, then the conversion will occur, but an error message will indicate lines where the format is inappropriate.

Once converted, the user invokes the FIS editor to include knowledge database rules in the file. Prerequisites are that a shell knowledge database file has been created and exists in the application memory of the FIS application. The user chooses the editor function within FIS. When entering the editor, the user selects the module editor menu item. The module editor then allows the user to add, delete, or change all of the information associated with modules and rules through two viable options, manual and automatic.

Manual Rule File Addition to FIS — The manual procedure requires that you choose the *add-module* command. FIS prompts the user for the module name, the failure rate, and the rules. The module name is evident. The failure rate is the normalized value of the System (in this case the AN/SQS-53B) component's mean time to failure. If this is not available, then a default value is used. Finally, the user may add individual rules as he would for adding small numbers of rules. If the rule files are already prepared, then the user provides a *NIL* as default when prompted by FIS to enter the rules for each module. This completes the setup of the file. Now the user adds the rules by invoking the *rule-file* command in the FIS editor menu. This requests the name of the file containing the rules and the index number of the module, in the FIS knowledge database, to which the rules will be added. The user should exercise caution to ensure that the index used matches the name of the module rules. Since only the index number - not the actual name - is used, confusion could occur. To obtain the proper index, the user may invoke the *show-module* command in the module editor. A module index must be used to access the rule files for each module. The process continues until all rules are added.

The following example illustrates the process involved in using the semiautomated/manual procedure for rule addition. In the following example these notes apply:

a. Each new step is boxed. This may be an entire display screen or just the relevant change from the previous screen. This method of presentation is used to conserve space and to clarify the actual action that occurred.

b. User responses are in bold text.

c. Explanatory comments are in italics between boxed text areas.

The example follows:

This menu results from selecting the editor selection at the top level of FIS.

```
INSPECT                 MISC.
- - - - -              - - - -
me: module-editor       q: quit
te: term-editor
pe: prec-editor
oe: order-editor

Enter command or ? --> me
```

The me option is used to enter the rules.

```
INSPECT          MODIFY           I/O        MISC.
- - - - -        - - - - - -      - - - -    - - - -
sm: show-module  am: add-module   r: read    li: lisp
sms: show-modules dm: del-module  w: write   q: quit
srs: show-rules  drs: del-rules
                 ar: add-rule
                 ef: edit-field
                 df: del-field
                 rf: rule-file
                 mm: modulemaker

Enter command or ? -->am
```

A module must first be added to the file.

```
Creating a MODULE entry
Enter NAME -->mod1
```

The user must enter the name of the module.

```
Enter FRATE -->.1
```

A failure rate is entered that may be a default or a result of normalizing the reliability information for the system. A number less than one is reasonable.

```
Enter CAUSAL-RULES -->nil
```

Entering nil for the causal rules completes the module information, otherwise manual rule entry would be required at this point.

```
INSPECT          MODIFY           I/O        MISC.
- - - - -        - - - - - -      - - - -    - - - -
sm: show-module  am: add-module   r: read    li: lisp
sms: show-modules dm: del-module  w: write   q: quit
srs: show-rules  drs: del-rules
                 ar: add-rule
                 ef: edit-field
                 df: del-field
                 rf: rule-file
                 mm: modulemaker

Enter command or ? -->sms
```

The sms command is selected so that the module names and their indices will be displayed.

The terminals and their indices are displayed below in groups of 14. When the "more" appears at the bottom of the list a ***return*** *will cause the next group to display. These are identified by the dots and the boxed screen indication.*

1.	POWER
2.	A26A1S10
3.	A26A1S8_END_TEST_SWITCH
4.	A26A1S9
5.	A10_DELAY_LINE
6.	A17_AMPL_CONTROL
7.	A1_DELAY_LINE
8.	A22_SIGN_CONTROL
9.	A23_ATOD
10.	A24_AMP
11.	A26_REFERENCE_CONTROL
12.	A29_DELAY_LINE
13.	A2_DELAY_LINE
14.	A30_DTOA
more	
.	
.	
.	
30.	MOD1

When the user is familiar with the indices he invokes the rf (rule-file) command.

Enter command or ? -->**rf**

Enter file name -->**"mod1"**

The name of the file is entered. It must be surrounded by double quotation marks.

Enter MODULE id -->**30**

The appropriate index is supplied.

Enter command or ? -->

The procedure can be repeated until all modules and rules have been entered.

Automated Rule File Addition to FIS — The automatic method of preparation uses the command *modulemaker*. When the user invokes this command, FIS prompts him for a file name whose contents include a list of all the module names separated by spaces, or on separate lines. To prepare this file in a simple way the UNIX command is used,

ls > modulenames

where the only contents of the directory are the module rule files whose names are the modules' names. If other files exist, the user can invoke a text editor to remove extraneous information from the modulenames file.

Following the prompting for the file name of the modules, the command prompts the user for an output file name. FIS adds a *.v* suffix to the name. When the function executes, it produces a file whose contents are exactly the same as the manual process, using a default failure rate of 0.1 (this can be edited later if necessary). From this point, the process of adding the rule is exactly the same as the semiautomated/manual process. All parts of the example for the semiautomated/manual-rule file addition procedure apply after the initial module addition. For the semiautomated/automated procedure the modulemaker command replaces it and the user is prompted to the name of the file that contains the module names. This process is named semiautomated because a significant level of human involvement is required to create the rules. The automated procedure, which minimizes human interaction, is described in the next section.

Automated Rule Addition

In the automated conversion, the data file converts directly from the rule database format to the format compatible with FIS. The listing of the program that performs this conversion is shown in Appendix F. This program constructs a complete *.v* file with the rule and test information, if the rule and test data files are provided as input. If just the rule part of the *.v* file must be created, then a dummy input file must be provided as the name of the testlist file. Conversely, if only the test information is to be entered, a dummy file is provided for the rule file input. Originally the procedures used in this program were separate programs, but they were joined to make a fast cohesive method for constructing the knowledge database contained in the *.v* file.

This conversion, when executed, first asks for the rule data file's name, then the name of the testlist file. Next the program prompts the user for the name of the output file. A *.v* suffix is added to the file name automatically. The conversion occurs resulting in a file whose name has a *.v* suffix. The result is a complete *.v* file in the correct format. This means that the rule and testlists are constructed in the proper place within the file, and *NIL*s are supplied for the lists not yet entered such as orders and preconditions. This is exactly the same as the result of the first method of conversion, except that the first method does not account for adding tests. In the first method a *NIL* is supplied in place of the testlist. If a dummy file is supplied for the name of the testlist file, the two produce exactly the same *.v* file. As in the semiautomated/automated rule addition procedure, the automatic conversion routine supplies a default failure rate of 0.1 for modules, which may later be edited as necessary. The *.v* file prepared in this manner, with rule and test information provided, is in a form that will permit a successful compilation to a working knowledge database.

Manual Rule Addition

The all manual method requires that FIS be loaded, a knowledge database *.v* file loaded, and the *knowledge-editor* command selected. Within the knowledge editor, the user should select the *module-editor* command and then select the command to add modules. FIS prompts the user to enter the module name and all the individual items composing the module. Each module is added separately. The manual entry of rules with the FIS editor is discussed in this report.

Testlist Conversion

The user may perform the conversion of the testlist manually, semiautomatically, and automatically in a manner similar to the conversion of the rulelist. Only the functional test information from the testlist file such as test name, test point, parameter, units, qualitative and quantitative values, the cost, type, setup and precondition information is used to construct the FIS testlist item.

As in the converted rule file, the parentheses enclose all elements of the file. Assuming a successful conversion without format errors, then a file is generated as the one below:

```
(Test-point (Test-name-1 Param
            Setup
            ((Qualval-1 ((Quant-1 Quant-2))
            (Qualval-2 ((Quant-0 Quant-1))
            (Qualval-3((Quant-2 Quant-3))))
            units
            test_type
            cost
            focal_module)
              (test-name-2
            ...
            ...
            ...)
).
```

The entire quantity is enclosed in parentheses to define, in LISP terms, that the information is a list of the tests. This quantity is then incorporated into the appropriate position within the *.v* file, replacing a *NIL* that held the place. The actual format and manner that results after the conversion depends upon the type of conversion used. Each provides a slightly different format because the whole process using that procedure must be considered. The processes are discussed in the following sections.

Automated Test Addition

The automated conversion of the testlist file is preferred in large databases to simplify and reduce the effort involved. Both the automated conversion process, contained in Appendix F and described in the automatic rule conversion, and the independent automatic testlist conversion, listed in Appendix H, accomplish the conversion automatically with similar amounts of effort. As stated for the rule data conversion, the automated conversion can incorporate both the rule and test information without further effort. The independent automatic testlist conversion deals only with the testlist file, and some slight amount of editing is required to arrive at the final format. The independent automatic testlist conversion is basically incorporated as a procedure within the automated conversion.

The independent automatic testlist conversion, when invoked, uses the testlist file as the input. The output of the conversion produces a file in the correct format as shown:

```
((Test-point (Test-name-1 Param
            Setup
            ((Qualval-1 ((Quant-1 Quant-2))
            (Qualval-2 ((Quant-0 Quant-1))
            (Qualval-3((Quant-2 Quant-3))))
            units
            test_type
            cost
            focal_module)
              (test-name-2
            ...
            ...
            ...)
)
(...)
(...)
(...)
).
```

This format has no connection, at this point, with the .v file. The user must append the file generated in the testlist conversion to the .v file, however this cannot be performed immediately. The testlist is a list within the .v file, just as the rulelist is a list. If it is not present, a *NIL* is there to hold its place. There are other items required in the .v file, and if any are not available, a *NIL* is supplied. At this point, after adding the rules, three *NIL*s are at the beginning of the .v file, and four *NIL*s at the end. The place for the testlist is immediately after the rules, as indicated:

NIL
NIL
NIL
(RULELIST)
(TESTLIST)
NIL
NIL
NIL.

To append the converted testlist the user must remove the *NIL*s at the end of the file. The user then appends the two files. The three remaining *NIL*s can then be added, or the remaining information to fill those *NIL* spaces may be supplied. The .v file, after this process, may be compiled.

The automated procedure is the preferred method of entering testlist information. The manual conversion and construction of the FIS testlist are preferred only for small amounts of data; for large amounts of data the automatic method is preferred. The fully manual or the semi-automated method become tedious for large amounts of data.

Manual Test Addition

The fully manual method requires that FIS be loaded, a knowledge database .v file loaded, and the *knowledge-editor* command selected. Within the knowledge editor the user should select the *terminal-editor* command. The user then selects the command to add tests. FIS prompts the user to enter the terminal identification number for all the individual items composing the test. Each test is added separately. This method is discussed further in this report.

Semiautomated Test Addition

The semiautomated method of test addition is an improvement over the manual method since it eliminates adding each component of every test. However, for knowledge databases with a large number of terminals it is still somewhat tedious. The program in Appendix I is executed to perform the semiautomated method of test addition. The testlist file becomes the input to the program. The user is prompted for the name of the output file, or if the output should be at the screen. The output file, named in this process, will contain only the list of errors and line numbers that occur during the conversion. The other form of output creates files named for each terminal in the list. The contents of these files are the tests associated with each of these terminals. The file contents are in the form shown below:

Testname, parameter, setup, specifications, units, test type, cost, focal-module;

An example of the contents of a file is:

t100 freq s1 ((OK) (hi) (lo)) hz perf 1.0 NIL
t200 freq s2 ((OK) (0 1) (bad ((-inf 0)(1 inf)))) hz diag 0.5 (mod1)
t101 freq s1 ((OK) (hi) (lo)) hz perf 0.3 NIL .

Having generated the test information files, the user may use the *test-file* command in the terminal editor. Again, as for the rule construction, the user must be certain that the index of the terminal is correct and matches the test information. By using the wrong index, the user enters the test information for one terminal into the data slot assigned to another terminal. This process continues until all test information has been entered. For large numbers

of terminals, as is generally the case when constructing an entire knowledge database, this method and the manual method take considerable time.

The following example is designed to illustrate the process involved in using the semiautomated procedure for test addition. In the following example these notes apply:

a. Each new step is boxed. This may be an entire display screen or just the relevant change from the previous screen. This method of presentation is used to conserve space and to clarify the actual action that occurred.

b. User responses are in bold text.

c. Explanatory comments are in italics between boxed text areas.

The example follows:

This menu results from selecting the editor selection at the top level of FIS.

INSPECT	MISC.
- - - - -	- - - -
me: module-editor	q: quit
te: term-editor	
pe: prec-editor	
oe: order-editor	
Enter command or ? --> **te**	

The te option is used to enter the tests.

INSPECT	MODIFY	I/O	MISC.
- - - - -	- - - -	- - - - -	- - - - -
stm: show-term	at: add-term	r: read	li: lisp
stms: show-terms	dt: del-term	w: write	q: quit
sts: show-tests	dts: del-tests		
	ats: add-test		
	ef: edit-field		
	df: del-field		
	tf: test-file		
Enter command or ? -->**at**			

A terminal must first be added to the file.

Creating a Terminal entry
Enter NAME -->**term1**

The user must enter the name of the terminal.

```
Enter TESTS -->nil
```

Entering nil for the TESTS completes the module information, otherwise manual rule entry would be required at this point.

```
INSPECT              MODIFY              I/O          MISC.
- - - - -           - - - - - -          - - - -      - - - -
stm: show-term       at: add-term        r: read      li: lisp
stms: show-terms     dt: del-term        w: write     q: quit
sts: show-tests      dts: del-tests
                     ats: add-test
                     ef: edit-field
                     df: del-field
                     tf: test-file

Enter command or ? -->stms
```

The stms command is selected so that the terminal names and their indices will be displayed.

The terminals and their indices are displayed below in groups of 14. When the "more" appears at the bottom of the list, a ***return*** *will cause the next group to display. These are identified by the dots and the boxed screen indication.*

```
1.      A78J5
2.      A20J5
3.      A1J4
4.      A1J5
5.      A1J1
6.      A1J3
7.      A1J7
8.      A11J6
9.      A1J6
10.     A1J8
11.     A77J4
12.     A76J3
13.     A1J9
14.     A10J9
more
.
.
.
30.     TERM1
```

When the user is familiar with the indices he invokes the tf (test-file) command.

```
Enter command or ? -->tf
```

```
Enter file name -->"term1"
```

The name of the file is entered. It must be surrounded by double quotation marks.

```
Enter MODULE id -->30
```

The appropriate index is supplied.

```
Enter command or ? -->
```

The procedure can be repeated until all modules and rules have been entered.

If errors have occurred in the conversion, they may actually be in the file format or may be a result of misaligned information in the file's columns. The conversion program is sensitive to column structure as well as format. If errors occur, they must be eliminated in order to obtain all of the test information for the appropriate terminal files. They can usually be eliminated by examining the data file at the point of the error and identified by the line number; then any column misalignment of the data must be corrected. This can be performed with a standard text editor such as vi. If errors persist, the user should review the format for data exceeding column allocations. However, if this occurs in only a few cases, the knowledge engineer may choose to enter the data with the manual editor. If the number of errors is more extensive, the user should revise the data file to meet the established format guidelines.

Testlist Conversion for Function Instructions

In the implementation of the TAS, a feature was added to FIS to display instructions, which required an addition to the data structure expressed in the *.v* and *.lisp* files. The additional data required are contained in two parts. The first is a test index that is appended to the *.lisp* file. The other data are in an *.instruction* file that contains the general instructions for all tests. The index information relates the special tests to the general instructions and contains information to specialize an instruction for a specific test. Each general instruction is stored as a multiple of a 512-byte block. The index identifies the record block in the *.instruction* file that relates to a specific test. FIS uses the information from both data parts to display a specific instruction.

Two conversion programs are used to create the two data items. One conversion program, listed in Appendix K, uses the instruction text file (exemplified in Appendix E) and converts it to an *.instruction* file used by TAS during instruction display. The input text file contains data in the following format:

Instruction_name: xxx
Instruction Text:
"general text" special_print_integer
"general text" special_print_integer "general text"
special_print_integer... .

In this format the *Instruction-name: xxx* identifies the name of a generalized instruction that the other conversion uses in the indexing process. The *Instruction Text:* is used by the first conversion to identify that the body of the instruction follows. *General text* indicates that this portion of text enclosed in quotation marks will appear each time this instruction is called in a FIS troubleshooting session. The *special_print_integer* is denoted by a *$* and an integer from 0 to 9. During instruction display, the specific value is supplied by the index data structure.

During the conversion, the *instruction_name*, the *general text* and *special_print_integers* are placed into an appropriate number of records within the instruction file. The integer 0 delimits the end of each instruction. Prompted by the large size of the input files, a linked list was used, thereby minimizing necessary memory. This instruction file exists separately from the knowledge database, since several knowledge databases use the same general instruction file. This was demonstrated in the application to the TAS hierarchy, where several knowledge databases all used the same general instructions.

The other conversion program, listed in Appendix J, creates the index for the instructions. The testlist file and the *.instruction* file (created with the first conversion) are the input files for this conversion. The format for the testlist file is the same as that required to create the FIS testlist. However for the conversion to create the index, two additional columns from the testlist file are recognized. Column 10 is the instruction name and column 11 relates additional text parameters. The LISP function generated, called *instr-list*, has the following format:

```
(defun instr-list ()
 '(
  (test_point-1
       (Parameter Instruction_name (Min Max) record_location [txt0]..[txt9]}
       (Parameter ...))
  )
(test_point-2 ...
  )
) .
```

In the output structure the following items are all used to fill in the *special_print_integer* items within the general instruction: *test_point*, *parameter*, *min*, *max*, and *text* items. The text parameters (*txt0* and *txt9* in the above example) are optional and appear only if provided by the input file. If the input file does not provide the values for *Min* and *Max*, then *(NIL NIL)* will be substituted for the minimum and maximum values. The *instruction_name* is used by the conversion process to correlate with the other input file, the .instruction file. The correlation with the .instruction file provides the *record_location* that is equivalent to the offset, in bytes, from the beginning of the data file, divided by the size of each instruction record block (512 bytes).

The index is arranged so that all test data with the same test point are grouped as sublists under that test point. This allows for a quicker look-up in the LISP searching routines.

Program flow and execution are very similar to the conversion of the testlist, except in the creation of the data file. After each data block is read it must be stored within the process memory rather than being written to an output file because of the ordering required for the data structure.

Accessing Instructions with the "Print-Instructions" Function During Execution

The command *Print-Instructions* was added to FIS specifically for the technician maintenance application. It is only used in the display of instructions during a maintenance application. It uses the LISP function *instr-list*, the index created in the previous section, to find the correct location of the instruction in the data file. This prints the generalized instruction along with any specified text parameters associated with the specific test point. The generalized instruction converts to a specific instruction that the technician can follow to perform the necessary test.

The *.instruction* file, described in the previous section, is opened by the *Print-Instructions* function whenever a test is presented to the user. The *Print-Instructions* function correlates the test with the instruction by calling the *instr-list* function and by using this function to index the appropriate record number. *Print-Instructions* moves the file pointer to the correct position in the *.instruction* file.

The file containing the instructions in record format should be placed in a file named *.instruction*, in the user's working directory. The purpose of having separate files is to reduce the requirements for RAM memory. These instructions are not elements that are actively used by FIS to operate properly, but rather an item accessed when necessary. Error checking is built into FIS to prevent the two files from being separated. The result is that only when instruction information is required, it is necessary to access the *.instruction* file.

Summary

The section "Knowledge Acquisition" described the data required by FIS. It also described the format that this data must conform to if automated conversion to a LISP format is to occur.

The process of converting to the LISP format is discussed in the section "Conversion of Knowledge Database from Human Data Management Format to LISP Compiled Database." The result is the formulation of a *.v* file. Also, the creation of the *.instruction* file that contains general instruction information, and the instruction index were discussed. At this point (with a *.v* file, an *.instruction* file, and an instruction index function, *inst-list*), the FIS

knowledge database is in a format that is not easily interpreted by humans, since it is in a LISP compatible format. FIS cannot use the data at this point either. The user must invoke the FIS compiler to create an executable Technician's Assister System for a specific UUT.

Before and after compilation it may become necessary to modify the knowledge database in the *.v* file format to correct minor problems. The process of editing and that of compilation are also discussed in this report.

Formulation and Modification of the FIS Compatible Knowledge Base by Using the Editor Function

Two approaches exist to formulate the FIS compatible rulebase. The first relies on manual entry of module, terminal, precondition, and order information in an interactive session with the FIS Editor. A more detailed description of this process can be found in Ref. 1. The second approach uses the FIS Editor only in a semiautomated mode, and only to create a shell for the appending of the rulebase whose LISP conversion has already been described. For large knowledge databases the use of the FIS editor is minimal. The majority of the effort is performed in the conversion process by using UNIX editing tools and functions.

Formulation of the FIS Rulebase Using the FIS Editor Exclusively

FIS has two types of editors that are supplied as standard features of FIS, a graphical editor and a knowledge database editor. No graphics were included in the TAS for the AN/SQS-53B application; therefore, this report does not describe the graphical editor.

Only the editor for the knowledge database is described. Once the user chooses the knowledge database editor from the main FIS menu and supplies an UUT name (the knowledge database for the UUT), then the user has four possible utilities that he can use in editing the knowledge database. The utilities and corresponding elements are module editor (for modules and rules), terminal editor (for test information), precondition editor (for precondition information), and order editor (for order information). All four editor utilities have a similar structure. The following sections briefly describe each structure.

Module Editor

Once the user selects the module editor, FIS presents the following menu:

INSPECT	MODIFY	I/O	MISC.
- - - - -	- - - - - -	- - - -	- - - -
sm: show-module	am: add-module	r: read	li: lisp
sms: show-modules	dm: del-module	w: write	q: quit
srs: show-rules	drs: del-rules		
	ar: add-rule		
	ef: edit-field		
	df: del-field		
	rf: rule-file		
	mm: modulemaker		

Inspect — Show-module: FIS prompts the user for the module index. When the user provides the index number (one to module total), all of the information for the module is then supplied; the name of the module, the failure rate, and the list of rules.

Show-modules: FIS presents a list of all the modules and their numeric index, in groups of 14-line lists. Pressing the *return* key presents the next 14 lines in the list.

Show-rules: FIS prompts the user for the module index. When the index is supplied, all the rules for that module are presented.

Modify — Add-module: This function allows the user to establish new modules. Once the user enters a module name, FIS prompts the user for the failure rate. After supplying the failure rate, FIS prompts the user for a list of rules. NIL is convenient to enter as the temporary value for the rule set. Actual rules can be entered later with the *add-rule* function or the *rule-file* function, for a group of rules.

Delete module: This function deletes all the information in the module, as well as the module name and its index.

Delete-rules: This function deletes a rule or rules from the module's information set.

Add-rule: This function adds a rule to the information for a specified module.

Edit-field: This function allows modification of defined fields.

Delete-field: This function deletes defined fields.

Rule-file: This function allows the user to add rules to a module's information set by supplying a file name to read the rules from. FIS prompts the user for the module's index number and the name of the file where the rules are stored.

Modulemaker: This function initializes the *.v* file to a form that initializes all of the modules with a default failure rate of 0.1 and no rules. FIS prompts the user for the file's name that contains the module names and the output file names for the UUT's *.v* file. The function supplies the *.v* suffix. This function is most efficiently used with the rulefile function.

I/O — Read: Read into memory a *.v* file

Write: This function writes changes to the *.v* file that is currently being edited.

Misc — LISP: This function allows the user to escape to LISP to execute LISP functions.

Quit: This function exits at the level where the element editor is selected.

Terminal Editor

Once the user selects the terminal editor, FIS presents the following menu:

INSPECT	MODIFY	I/O	MISC.
- - - - -	- - - - - -	- - - -	- - - -
stm: show-term	at: add-term	r: read	li: lisp
stms: show-terms	dt: del-term	w: write	q: quit
sts: show-tests	dts: del-tests		
	ats: add-test		
	ef: edit-field		
	df: del-field		
	tf: test-file		

The terminal editor is similar to the module editor. The function for I/O and MISC. are identical. The other functions are similar.

Inspection — Show-term: FIS prompts the user for the terminal index. When the user supplies the index number, from 1 to terminal total, FIS displays all the information for that terminal such as the terminal name, and the test (test name, parameter, setup, qualitative and quantitative values, units, type (performance or diagnostic), cost, and focal module).

Show-terms: FIS presents a list of all the terminals and their numeric index, in groups of 14-line lists.

Show-tests: FIS prompts the user for the terminal index. When the index is supplied all the tests for that terminal are presented.

Modify — Add-term: This function allows the user to establish tests for new terminals. Once the user enters a terminal name, FIS prompts him for the tests. This must be supplied in precise format without prompting. For users unfamiliar with the test format required, it is more efficient to enter tests individually with the add-test function, where the user is prompted for information. If *add-test* is to be used, then *NIL* should be entered when prompted for tests.

Del-term: This function deletes the terminal and all information associated with the terminal, including the index.

Del-tests: This function deletes the test information for a given terminal.

Add-tests: This function adds a test to the information for a specified terminal. FIS prompts the user for each item of information.

Edit-fields: This function allows modification of defined fields.

Del-field: This function allows deletion of defined fields.

Test-file: This function allows the user to add tests to a terminal's information set by supplying a name to the file to which the tests will be read. The user is prompted for the terminal's index number and the name of the file where the tests for that terminal are stored.

Precondition Editor

Once the user selects the precondition editor, FIS presents the following menu to the user:

INSPECT	MODIFY	I/O	MISC.
- - - - -	- - - - - -	- - - -	- - - -
sp: show-prec	ap: add-prec	r: read	li: lisp
sps: show-precs	dp: del-prec	w: write	q: quit
	ef: edit-field		
	df: del-field		

The precondition defines states which must exist to perform groups of tests. Note the similarity with the module editor. The functions for I/O and MISC. are identical. The other functions are similar in form.

Inspection — Show-prec: FIS prompts the user for the precondition index. When the user supplies the index number, from 1 to precondition total, FIS displays all of the information for that precondition — the label and definition.

Show-precs: FIS presents a list of all preconditions and their numeric index, in groups of 14-line lists.

Modify — Add-prec: This function allows the user to establish the preconditions for the UUT. First, FIS prompts the user for a label; this is the precondition name. Once the label is provided, the user is prompted for a definition that must be in the form of a LISP list.

Del-prec: This function deletes the precondition and the definition of the precondition.

Edit-field: This function allows modifications of defined fields.

Del-field: This function allows deletion of defined fields.

Order Editor

Once the user selects the order editor, FIS presents the following menu to the user:

INSPECT	MODIFY	I/O	MISC.
- - - - -	- - - - - -	- - - -	- - - -
so: show-order	ao: add-order	r: read	li: lisp
sos: show-orders	do: del-order	w: write	q: quit
	ef: edit-field		
	df: del-field		

The order defines the requirement for test procedures. The similarity with the module editor is noticeable. The functions for I/O and MISC. are identical. The other functions are similar.

Inspection — Show-order: FIS prompts the user for the order index. When the user enters the index number, from 1 to order total, FIS displays all the information for that order — the successor and predecessor, or predecessors.

Show-orders: This function displays a list of all the orders and their numeric index.

Modify — Add-order: This function allows the user to establish the order of testing required when testing the UUT. The user is first prompted for a successor; this is the name of the order. Then FIS prompts the user for a single predecessor, or a list of predecessors in a LISP list.

Del-order: This function deletes the order.

Edit-field: This function allows modifications of defined fields.

Del-field: This function allows deletion of defined fields.

Formation of the Knowledge Base for Large UUTs

The formation of the knowledge database for the AN/SQS-53B was performed as already described by using the data in text format and individually converting each item to a modified format. In this manner, rules and tests were added to the knowledge database by module and terminal name respectively. The preconditions and orders were each formulated in the LISP format and appended separately to the knowledge database. Since the capability was already available (through the early stages of development) to transform data to a knowledge database format, the individual components were combined into a single program, as listed in Appendix F. In this case, by invoking this program the user is prompted for all the names of each type of information. Thus, the user is asked for files that contain the rule, test, precondition, and order information. The function performs all of the required conversions and produces a complete knowledge database. If the data files contain data that are not in the correct format, a message will appear indicating that an error has occurred in reading the data file. A line number identifies the error location. Errors indicate that a particular line in the data file has data that do not conform to the format defined.

To correct errors, one must examine the data file and modify the data to conform to the format. Upon modifying the data file, the function to create the knowledge database should be executed again. If errors that are not readily explained persist, modification and reforming may be tried again. If the data that cause the errors are not extensive, the user may choose to enter it manually by using one of the knowledge database editors. Assuming that the user is not aware of the intricacies of the data format, it may be appropriate to use the FIS editor.

Once the knowledge database of the UUT is complete, a function that converts the information back to the data format is also available (the program listing is found in Appendix L). Even if no data files had existed previously, and all the knowledge had been entered with the knowledge editor, the data files can be created with this program. The purpose of maintaining or creating the data file is to ensure the integrity of the data, through data management of a format that a user can easily view. In this manner, changes made after the compilation and validation can be incorporated into the database. Similarly, if changes occur as a result of engineering changes made to the UUT, the data changes can more easily be traced from data file to knowledge file. Ultimately, information in the knowledge database relates directly back to the circuit topology of the UUT. The data files exist to ensure the integrity of the data for the TAS.

Although the programs discussed in this section, and listed in Appendices F and L, are written in the C programming language and not presently part of FIS, they will be incorporated as functions within the knowledge editor. In the next updated release the functionality of the program in Appendix F will be contained in a single function within FIS called *ckb: create-knowledge-base.* Also in the updated release, the functionality of the program in Appendix L will be contained in a FIS function called *ctd: convert-to-data.* The addition of these functions will consolidate the programs used to convert information in a data format to information in the knowledge database format of FIS.

Compilation

After assembling all the component parts of the knowledge database, they must be compiled into a format that FIS can execute. To compile the knowledge database in the *.v* format into a functional *.lisp* file, the process begins by entering FIS and choosing the compiler command. The FIS compiler will prompt the user for the file name. This file name is entered without suffixes, i.e., *.v*. FIS then compiles the information into a file with a *.lisp* suffix. FIS will reply with several messages to indicate that it is working, and several diagnostic messages will appear. The messages are to indicate to the user that simple diagnostic precompilation tests are being performed. These tests check the syntax of the *.v* file contents. Errors in syntax can be symptomatic of more critical errors within the file. The compiler provides a message that indicates testing of each item. Primarily, FIS informs the user that it is checking the syntax of modules and terminals. Duplicate module and terminal names are indicated during the initial check. After the initial check, a full diagnostic report is provided with the information that triggered the warning flag.

The following is a list of the diagnostic messages that FIS presents to the user, and an explanation of each. The actual message is presented in italics.

a. *The rules contain the following modules that have not yet been defined* — This diagnostic provides a list of module names. The module names are those that appear as causes in the rules. It is required that if a module is a cause, it should be defined as a module and must contain a set of rules. The solution is to create a module and add the rules that apply. Otherwise, if the indicated item is not a module, a triple should replace the atom in the cause, of the rule triggering the diagnostic message.

b. *The following modules are not mentioned in the rules* — This diagnostic provides a list of module names. The module names listed are structured properly; however, they do not possess a rule in their rule set, nor does a rule appear elsewhere in the knowledge database that identifies the module as a cause. This condition does not allow for closure of the knowledge. There must be at least one rule to identify each module as a possible cause for bad output signals from the module. The knowledge engineer must create a rule to identify the module as a cause to a valid effect.

c. *The rules contain the following terminals that have not yet been defined* — This diagnostic provides a list of terminal names. The terminal names in the list identify those terminals that have no tests. This can be a result of no test existing for that terminal, or a terminal that has a test but has not been identified in the terminal list. The second cause is a condition that should be corrected by adding a terminal and a test. The first condition is not necessary to correct since it is not required that all terminals have tests. In the application, terminals with *J* in the name are to indicate testable point. Other terminals do not require tests.

d. *The following terminals are not mentioned in the rules* — This diagnostic provides a list of terminals. The terminal names in the list identify conditions where terminals exist with no relation to the rule. This can be a symptom of an extraneous terminal test or an error in the rules. If the terminal test is extraneous it can be eliminated. If it is a result of an error in the rules, then the rules must be created or modified to use the terminal as a test point for knowledge database.

e. *The following labels are used for both terminals and modules* — This diagnostic provides a list of names that have dual definitions as modules and terminals. This state should not exist, and those names with dual definitions should be modified in all appropriate places in the rule set to read either as a module or a terminal. If not corrected, linkage of the knowledge database elements could be affected.

f. *A rule's effect must be a triple of the form (terminal parameter qualitatives-value). The following atomic effects were found* -— This diagnostic provides a list of improper effects that exist in rules. The solution is to provide a triple of the correct form.

g. *The following terminals have no performance or diagnostic tests* — This diagnostic provides a list of terminals. One of the items required for the definition of a terminal test is an indication of whether it is a performance or diagnostic test. Performance tests are performed first to determine functional abnormalities. Diagnostic tests are performed to isolate and correct a functional abnormality.

h. *The following modules were defined twice* -— This diagnostic provides a list of modules. This indicates that a module and rule occur in two different locations within the knowledge database. This condition provides contradictory information for the knowledge database, and linkage of the knowledge database elements is threatened. The solution is to move the rules from one location to the other and eliminate the second definition of the module.

i. *The following terminals are defined twice* — This diagnostic produces a list of terminals. The diagnostic points out effects similar to the case of diagnostic Item h for modules. The information, by duplicating a definition of the terminal, confuses the information in the knowledge database. The solution is to remove the duplicate definition and place the additional test information into the single terminal definition.

Miscellaneous

As the FIS compiler produces all of the above diagnostics, additional diagnostics exist that may result from an error that is handled by the LISP debugger. In general, this type of error is a result of a breach in the LISP format. Usually an error will occur that will be a segmentation violation or some very similar error. These types of errors are not predictable, so it is impossible to define all the possible error messages that may occur. The user should reference the SUN LISP manual for an explanation of errors that result in entry to the LISP debugger. In the case of knowledge database compilation, the majority of errors result from misplaced parentheses. This condition can be diagnosed and corrected by removing, replacing, or repositioning the parentheses.

The result of all but miscellaneous diagnostics is a warning, during compilation, that syntax errors exist. FIS prompts the user for an answer as to whether compilation should continue. If the user answers "no," compilation will stop. If the user answers "yes," compilation will continue. Additional messages will appear to indicate that the compiler is still working. Next, the compiler asks the user whether to remove the rules or not. Removing the rules saves some storage space, but disables the ability to trace the causal network. Whether the rules are removed or not, the next indication is that the compiled version is being written to a file. The file writes to a file with the same

UUT name as the *.v* file, and the suffix now becomes *.lisp* to indicate that compilation has occurred. Compilation ends with a return to the main menu.

Addition of Instructions

Once compilation of the file has occurred, the process used to create the instructions is invoked to create a viable *.lisp* file incorporating the instruction information used in the TAS/FIS. This is performed by creating the instruction file *.instruction* and appending the instruction index to the bottom of the *.lisp* file. Without the index added and the *.instruction* file existing, an error warning will occur. The *.lisp* file is complete for the TAS/FIS with the instruction index appended to it.

VERIFICATION AND VALIDATION

Although the diagnostics within the compiler are capable of detecting obvious faults in the knowledge database, they are incapable of detecting more subtle errors that produce major impacts. A major impact of an error occurs when the knowledge database appears to act properly in a procedural sense, but has errors in the rule continuity, thus preventing proper functionality. That means that the system will not converge to a system fault regardless of the amount of testing performed.

To verify and validate the data, FIS includes software to perform some of the testing of the system. There are basically three forms of verification performed. The first is to examine the connectivity of the data by examining the rules and ambiguity set information. The second is to use a logic simulator, resident in FIS, to test for convergence. And finally, if the actual hardware system is available, the technician's assister system should be validated against tests made on the system. This will determine whether the knowledge database converges by using real test data. The following sections will explain each of these methods.

Verification of Data Connectivity

Two types of verifiable data exist. Each provides its own diagnostic information. Verification tests are performed on the rules and ambiguity sets. At the main FIS menu the user chooses the *Hard-copy* option. The menu for hard copy is as follows,

Programs	
1.	Generate graphical description of UUT
2.	Generate ambiguity set output
3.	Generate cost code information output
4.	Generate rule set output
5.	Generate history list
6.	Exit

The user selects the desired option. The selected option from this menu results in data files suitable for printing. Since the data is extensive printing is preferred. The printed data is used for verification analysis. The only items used for verification are Item 2 (generates ambiguity set output) and Item 4 (generates rule set output).

Verification of Rule Continuity

This verification tool, as well as the others mentioned, are implemented after compilation. The user uses the output from this tool to examine the continuity of the rules. From this information the knowledge engineer may determine the continuity from any point in an upstream and downstream direction. An example of the form is seen in Appendix M. By using this data, the knowledge engineer logically traces the connections in a point-to-point manner through the knowledge database and determines whether they conform to the relationship represented in the system schematics.

Verification of the Ambiguity Sets

The verification tool (an example output is provided in Appendix N) examines the ambiguity sets arising as a result of all tests. It indicates for each test the modules that could be at fault if a failure occurs. This information is important since it allows the knowledge engineer to examine each ambiguity set. By doing this he is able to determine whether modules are correctly identified as suspects. Also, blank spaces demarcate places where tests exist but the knowledge database does not properly use them. This results in modules being absent from an ambiguity set for these tests. The knowledge engineer must then determine which rules are missing that would use these tests, or if the tests are useless. In general, the assumption is that rules should be added that would use these tests. The reason for this assumption is that the equipment design (for the AN/SQS-53B) is stable. The knowledge database constructed from this design must use a limited number of test points provided by the designer of the system. Since a limited number of test points exist, and consequently tests, it is advantageous for the knowledge engineer to use all of the test points available. This optimizes the inclusion of test points, and allows efficient fault isolation.

At present, the ambiguity set list must be examined in conjunction with the test list, since tests without ambiguity sets are not identified. Only a double separation line appears to demarcate these tests. By using the testlist, the knowledge engineer can identify the test and verify its utility. This is possible since the ambiguity set command lexically orders the occurrence of tests. After identifying the test, the knowledge engineer must determine whether the rules properly use them. Otherwise, the knowledge engineer should remove the knowledge database representation of the test terminal.

Validation of the Knowledge Database

Once the knowledge database has been constructed, compiled, and verified for consistency, the only remaining step is to validate the operation. That means that the computer based technician's assister should isolate faults and assist a technician in maintaining the system. Two assertions in the preceding statement should be noted. First, the AI system must be able to isolate faults. This entails singling out the module that is bad in a system, or if multiple faults exist, finding all the failures. Secondly, the technician must find the AI system useful. It must allow the technician to isolate faults quickly (faster than he may have done on his own). It must also act in a manner that is consistent with the technician's training and reasoning ability. Both of these assertions are important to the creation of a viable expert system for fault isolation.

Presently, two methods of conducting the validation process exist. First, FIS has a fault simulation capability that allows the user to set faults and then go through a troubleshooting session. The second method is to test the system in an operational environment to determine if the computer based technician's assister acts properly and efficiently in isolating faults.

Validation Through Fault Simulation

FIS has two modes of simulating a troubleshooting session. Each has its advantages, and both should be performed for all or most modules. Both methods must be accessed by invoking the demonstration mode from the main menu. Once invoked, the user loads the appropriate UUT's knowledge database, the *.lisp* file. After loading, the command used to set faults should be chosen. Setting a fault allows the knowledge engineer to set a fault to a module or node (a local effect) and further allows him to simulate the action of testing. The control should then be returned to the main demonstration menu. The following sections explain the automatic and manual modes.

Automatic FIS Validation

The user begins the fault simulation process by first initializing the simulation with a fault condition. The fault condition is used by the simulator to trace the logical result of simulated tests. The knowledge engineer enters the fault condition by invoking the *set-fault* function. Once the *set-fault* function is entered the user is prompted for the type of fault to set and the name of that module or effect. This menu is displayed below:

```
Fault Options
-------------
c: clear
m: module
i: imm-eff
s: show
q: quit

Enter fault option?-->
```

A list of faults can also be entered to simulate multiple fault conditions, if either the "module" or "imm-eff" (immediate effect) options are selected. If the "module" option is selected then the module names are displayed as seen.

```
Modules: (POWER A26A1S10 A26A1S8_END_TEST_SWITCH A26A1S9 A10_DELAY_LINE
A17_AMPL_CONTROL A1_DELAY_LINE A22_SIGN_CONTROL A23_ATOD A24_AMP
A26_REFERENCE_CONTROL A29_DELAY_LINE A2_DELAY_LINE A30_DTOA A31_DTOA A32_DTOA
A33_DTOA A3_DELAY_LINE A4_DELAY_LINE A5_DELAY_LINE A7_DELAY_LINE A9_DELAY_LINE
CORRELATOR_REF DOWNSTREAM_SONAR_UNITS MOD_REFER MUX_REF PMFL TIMING
UPSTREAM_SONAR_UNITS)



Enter module to be faulted -->
```

If the "imm-eff" option is selected then the list of terminals is presented to the user:

```
Terminals: (A78J5 A20J5 A1J4 A1J5 A1J1 A1J3 A1J7 A11J6 A1J6 A1J8 A77J4 A76J3 A1J9 A10J9 A10J1
A10J5 A10J4 A10J8 A10J6 A10J7 A10J3 A11J5 A11J2 A10J2 A21J13 A21J14 A11J3 A2J6 A11J16 A7J4
A11J14 A24J2 A20J8 A11J12 A11J10 A11J11 A20J6 A24J1 A24J5 A11J15 A11J18 A21J8 A29J8 A29J6
A29J7 A3J6 A3J8 A3J7 A30J4 A30J1 A30J2 A30J3 A20J2 A79J6 A77J7 A31J4 A31J1 A31J2 A31J3 A20J3
A32J4 A32J1 A32J2 A32J3 A20J4 A33J4 A33J1 A33J2 A33J3 A4J4 A4J9 A4J5 A4J1 A4J3 A4J6 A4J7 A4J8
A5J6 A5J8 A7J9 A7J5 A7J1 A7J3 A7J7 A7J6 A7J8 A9J8 A13J1 A11J4 A11J8 A11J9 A11J7 A58J7 A59J7
A60J7 A61J7 A24J4 A43J1 A43J3 A70J7 A75J2 A70J5 A71J1 A70J3 A74J1 A20J9 A20J11 A21J11 A20J13
A21J5 A26A1DS8 PMFL_J A42J5 A45J5 A46J5 A47J5 A48J5 A45J4 A46J4 A47J4 A48J4 A45J3 A46J3
A47J3 A48J3 A45J2 A46J2 A47J2 A48J2 A59J1 DISPLAY A98J2 A98J1 A42J3 A26A1J9 A26A1J10)


Select terminal or (q)uit -->
```

Upon setting the fault, the function is exited by entering "quit" and the simulator function is selected from the demonstration menu. After entering the simulation, the system automatically generates all of the tests that would be performed and the anticipated results. This is done by using default logic; special logic reasoning can be supplied in file form when the simulator queries whether default reasoning should be used, as seen in the following example:

The simulator is selected from the demonstration menu.

```
Enter command or ? -->s
```

Default reasoning is selected to use in the simulation.

```
Use default simulator reasoning (y or n)?: y
```

FIS performs the logical sequence of test and supplied results consistent with the fault being simulated.

```
Making test = (G-IF PWR S1) with result(s) = (HI LOW).
The results (HI LOW) are ambiguous - choosing HI
Making test = (M-IF PWR S1) with result(s) = (HI LOW).
The results (HI LOW) are ambiguous - choosing HI
Making test = (G-IF PHASE S1) with result(s) = (LOW HI).
The results (LOW HI) are ambiguous - choosing LOW
Making test = (SORT-CONT LOGIC S1) with result(s) = (BAD).
Making test = (F-AGC LOGIC S1) with result(s) = (BAD).
No best test available
```

The simulation continues until a termination is reached.

The path generated from the simulation is presented to the user, the simulator function is terminated, and the user is returned to the main demonstration menu to request an additional command.

```
   TRAIL
   -----
   (G-IF PWR S1 HI)
   (M-IF PWR S1 HI)
   (G-IF PHASE S1 LOW)
   (SORT-CONT LOGIC S1 BAD)
   (F-AGC LOGIC S1 BAD)



Enter command or ? -->
```

This process generates the most efficient path for testing the system, assuming that all system faults would display physical characteristics that are consistent with the logical representation. The simulation anticipates this consistency. This mode is useful in determining that each module can be fault isolated. It is a useful test of convergence.

FIS Assisted Manual Validation

The FIS assisted manual validation is achieved by setting a fault using the above process and performing a manual troubleshooting session or sessions. In this mode, the knowledge engineer sets the module fault. As the FIS system suggests each test, it displays a logically simulated result that would appear during that test. The person performing the simulation in this manner can then examine the probabilities and the ambiguity sets after each test. This mode gives the sense of an actual troubleshooting session and the process involved; it indicates the times involved in the troubleshooting sessions and the correlation with the automatic FIS validation mode. With this information, modifications can be made to the TAS. An example of the display is shown below. This is the standard best test display, except that the *SIM* column contains the simulation value that will be logically consistent with the rules in the knowledge database.

TEST	QUALVAL	SPEC	SIM
- - - - -	- - - - - -	- - - -	- - - -
DISPLAY	OK	NIL	(BAD)
CP_BEAM_4_WAVEFORM	BAD	NIL	
S1			
SS1			
DISP_4_WAVE			
Back to (t)erminals, (p)arameters or (s)etups, (q)uit or select measured value/qualval -->			

Two items should be considered when performing the manual simulation. First, all modules that are not faulted receive a certification factor that they are good. Conversely, the probability of failure increases only for the modules in the ambiguity set of the faulty module. Secondly, after each failed test, the ambiguity set should be examined to determine whether the testing is resulting in convergence. The knowledge engineer should examine whether the "faulted" module is a member of the ambiguity set. As testing increases during the simulation, the size of the ambiguity set should decrease, but should continue to contain the "faulted" module.

A convenient way to record the manual simulation process is by opening a UNIX script file. This can be done while still in the UNIX shell, before entering FIS. From that moment until the user closes the script by entering *control D* or "exit," all information that passes through the terminal will be recorded to a file. The file can then be further examined for weaknesses in the process of fault isolation.

Validation Through Field Testing

Once the knowledge engineer is satisfied with the performance, field testing should be performed. This allows technicians familiar with the hardware and troubleshooting to examine the operation of the AI system.

Testbed Testing

The first test in actual field exposure should be at the factory, testbed, or other site where hardware engineers are present. The testing of the AN/SQS-53B, Unit 26, technician's assister was performed at the Naval Underwater Systems Center (New London, CT). The purpose of testbed tests is to obtain information on how the sonar system works, its BIT capabilities, and the troubleshooting process. The knowledge engineer obtains the best exposure by reviewing the BIT to determine its effectiveness. This allows him to determine how well the expert system utilizes the available information. It is often difficult for the knowledge engineer to realize the capabilities of BIT from the technical description.

In the validation process for the sonar system technician's assister, for example, it was determined that an indicator light for the correlator functions as a more powerful test than was originally anticipated. Although the TAS acted properly, the power of this single test enhanced its operation.

Full Field Test and Technician Exposure

The final test in the validation process is to perform a full field test with technicians using the TAS. One purpose of the TAS is to aid inexperienced technicians in the entire troubleshooting process. Another is to aid experienced technicians in isolating difficult faults. The technician's assister system needs to be tested in situations with both types of technicians. This determines the accuracy of fault isolation, the veracity of the fault isolation process, and the ease of human interaction. These should be evaluated during the field testing process, with the greatest emphasis placed on the human interaction with the computer. The technician's assister system should aid the technician, not frustrate him with the process, the interaction, the communication, or the language. It may be useful to compare the operation of the system with an expert technician troubleshooting a problem. This should

provide a baseline for performance and not as an absolute evaluation criterion, since the system will not compete with experts, but assist those without expertise.

The prototype expert system developed for the AN/SQS-53B, Unit 26 has not yet been evaluated and validated in a field testing environment. This form of validation is to be performed independently by the Naval Sea Systems Command. The field validation may be conducted at an AN/SQS-53B technician training center. The results from the independent validation process may influence further developments of the system as they become available to NRL.

SUMMARY

The process of creating a knowledge database that together with the FIS shell forms an artificial intelligence based technician's assister system takes several steps. The steps are knowledge acquisition — obtaining the cause-effect rules, tests and testing instruction — from schematics or design information. Information obtained is then converted from human data management form to a LISP language format. Compilation follows by placing the knowledge database in a form that the computer can operate on efficiently. Verification then ensures the integrity of the knowledge database. Finally, validation assures the developer that the system constructed from the knowledge database acts properly. At each stage there is recourse to revert to previous steps to make knowledge database modifications.

The process of going from knowledge acquisition to a technician's assister system may require several iterations to achieve a valuable expert system. The development of the TAS knowledge database stressed the importance of data management. The transition from a data management form to the form used in the executable expert system was extensively developed and simplified to expedite the process of knowledge database creation. All information for the TAS should be available in the knowledge database. Any modifications made to the prime system should also be made in the knowledge database to maintain the configuration. If this is done, a new version for the expert system can be generated at any time.

The appendices contain samples of the development of the sonar system's knowledge database. They are meant to be a guide to the structure and format required for configuration management. They are also intended to display the format required for conversion to a knowledge database the computer can understand.

As a final note, the sonar system was much larger than any other system previously constructed with FIS. By increasing the size of the system modeled, speed degradation was observed. As a result, a modified architecture (TAS) was developed to improve performance. This architecture consisted of breaking the sonar system up into functional units that were much smaller, and as such, allowed fault isolation speeds comparable to smaller systems. The parts of the system were then organized by a top level information gathering system. The top level made bulk acquisition of indicator light and BIT test information possible. Its primary purpose was to isolate to the functional area. At the same time, the top level then selected which technician's assister knowledge database to examine for faults. Subsequently, it loaded the knowledge database for that functional unit in the system, and acted as a typical FIS technician's assister [4]. The entire system was named the Technician's Assister System, which was composed of two layers of software functionality. The expert system to perform the analysis to the functional level was named the local area expert (LAE). The analysis to the module level was performed with the software layer based on FIS. This layer was named the fault isolation layer (FIL), a name chosen to indicate that the software was based on FIS and also to indicate that enhancements were made to the basic FIS shell to accommodate the application. It is generally believed that large systems with BIT tests can be adequately handled in this manner. The testing architecture for entire systems could be developed by incorporating these techniques of knowledge acquisition and validation.

REFERENCES

1. F. Pipitone, K. A DeJong, and W. Spears, "An Artificial Intelligence Approach to Analog Systems Diagnosis," NRL Report 9219, Sept. 1989.

2. Servicing Diagrams Manual, Receive Subsystem for Sonar Detecting - Ranging Set, AN/SQS-53B(V) 4(1), SE313-TP-MMC-040, 4-39 & 4-53, Department of the Navy-Naval Sea Systems Command and General Electric Company-Avionics and Electronic Systems Division, DATOM Contract #N00024-84-C-6232,Oct. 1985.

3. Maintenance Data Diagrams Manual for Sonar Detecting - Ranging Set, AN/SQS-53B(V) 7(2), SE313-TP-MMC-090, 3-193, Department of the Navy-Naval Sea Systems Command and General Electric Company-Avionics and Electronic Systems Division, DATOM Contract #N00024-84-C-6232, Oct. 1985.

4. J. A. Molnar and G. Moss, "A Hierarchical Artificial Intelligence Maintenance Advisor," Proceedings of American Defense Preparedness Association Symposium and Workshops on Artificial Intelligence Applications for Military Logistics, Williamsburg, VA, March, 1990.

GLOSSARY

Ambiguity set - A set of all modules that could be faulty as a result of a failed test. This is based on the set of causal rules that are used to describe the UUT.

Abnormality - The manner in which a physical parameter at a terminal deviates from its specified value. Also referred to as the state of the test [1].

ATE - Automatic Test Equipment is a system characterized by a computer controller and an assortment of test making equipment. They are guided by software to assess the functionality of a unit under test, without human intervention. The ATE is a separate system developed specifically for the purpose of performing diagnostic tests. It is separate from the unit under test, but connected to the unit under test with a test fixture.

Best test - A standard test selected by FIS automatically. Such a test is found by a combination of heuristic screening of possible tests and maximization of the expected information gain divided by the cost (primarily in time) of making the test [1].

BIT - Built-in-test is the subsystem of a system; its specific function is to perform diagnostic tests to assess the functionality of the system without connecting separate test equipment to the system. The BIT performs testing either while the system is operating in its normal capacity, or in a mode in which only controlled testing is occurring. The BIT in either case does not interfere with the designed functionality of the system; its only function is to perform system diagnostic analysis.

Cause - The upstream portion of a causal rule. It can represent an abnormality at a terminal — in which case it has the same form as an effect — or it can be the name of the module. The latter is used in a rule asserting that a module can be faulty so as to cause some problem (that problem is the effect of the rule) [1].

Causal rule - This is a qualitative description of a causal relationship between two terminals of a module, or between a module and one of its terminals. Each rule has the form *(If <precondition> then <cause> <type> <effect>)*. Each rule is associated with a particular module whose behavior it partially describes. TA/FIS does not use the type. The cause is either the name of a module or an abnormality of a physical parameter at a terminal. The effect is an abnormality of a physical parameter at a terminal. The precondition (optional and often absent) is a binary function of the current state of FIS. For example, in the case of a multiplexer, we might have a rule (in English paraphrase) "If the select line is 'logic high' then input2 frequency high causes output frequency high." The

precondition here enables FIS to not follow the path from cause to effect if the multiplexer is currently believed to be switched off with respect to input2 [1].

Connectivity - This is the logical interrelationship of the rules that describe the physical relationship of the electronic system being modeled by the FIS knowledge database. Rules should exist which link together to create the same logical path that is seen in the physical interconnections of a circuit.

Convergence - The ability of the of FIS to use the knowledge database to correctly isolate faults. If FIS cannot converge on any faulty module this indicates a format error or an error in the logical connectivity of the rules within the knowledge database.

Downstream faults - A fault in a module whose test results rely upon the integrity of the modules before it in the signal path.

Effect - This the downstream part of a causal rule. It has the form *(<terminal> <parameter> <abnormality>)*. An example is *(input1 voltage low)* [1].

Entropy - This is the sum of p*log(1/p) over the fault states of the unit under test. P is the probability of a state. The sum is not computed directly, but by an efficient polynomial time algorithm [1].

FIS - Fault Isolation System is an AI software system designed to provide system diagnostic information in testing applications, based upon heuristics and entropy calculations. A knowledge database is required for the system to operate upon. FIS also contains editing functions available to assist the knowledge engineer in the creation of a syntactically correct knowledge database. Assorted functions are also available to assist the knowledge engineer in validating the accuracy of the knowledge database. Primarily FIS provides an inference engine to assist humans in performing the most efficient diagnostics on system hardware for which a knowledge database exists.

Module - This is a replaceable component in a UUT. FIS describes a UUT as a fixed set of modules. Each module has various data associated with it after UUT knowledge acquisition. This includes a set of causal rules, an a-priori relative probability, and a replacement cost [1].

Pseudonode - A logical construct used in the causal rules of the knowledge database to present the capability of analyzing the connection between modules and the backplane of a system. These nodes do not exist, nor can they be tested directly. Instead, their existence is defined as all points between any two real terminals, not at any single point. As with other terminals, they are part of some module, usually the backplane module. They are used in the causal rules as part of a triple, with a parameter and state as the other two elements of the triple.

Pseudoterminal - A logical construct used in the causal rules to reduce the total number of rules. The pseudoterminal is defined as existing within a real module, and always connected to two real modules also defined in the same module: an output and an input terminal to the module. The goal of pseudoterminal is to provide an additive factor for multiple interconnections between input and output terminals within a module. This is an improvement over the multiplicative factor that would exist if the input and output terminals were directly connected.

Qualitative values - The coarse values given to a parameter at a given terminal. The abnormality states are defined for it, plus the special value of "ok." A typical set of qualitative values is *(ok low high)*.

Quantitative values - The numerical equivalent of the qualitative values. FIS equates ranges of numerical values to the qualitative values.

Rulebase - This is the database that contains all of the causal rules for the S knowledge database.

Rulelist - This is another name for the rulebase.

Segmentation violation - An error occurring in LISP that usually results from a misplaced parenthesis.

Sonar system - The hardware consisting of several different functional units organized and controlled to perform the single task of monitoring perturbations in liquids. Primarily these systems are used on ships to detect and identify objects in water and monitor their movements. Acoustic waves propagating through the water are processed to provide information on their emanation or reflection point. The AN/SQS-53B is a specific sonar system used in this research.

TA/FIS - Technician Assister/Fault Isolation System is an enhanced FIS version developed to provide an Artificial Intelligence System to assist technicians in the diagnostic testing of the AN/SQS-53B. Primarily TA/FIS is the model-based part of the two level system hierarchy. TA/FIS (or FIL) provides isolation to the module level, while the Local Area Expert provides isolation to the functional area. Some enhancements found in TA/FIS are textual presentation of test instructions, recommendation of replacement, histogram of module probabilities, and a touch interface for communicating with the computer.

Terminal - This is a connection between two modules. Some terminals are test points. Terminals can represent any causal conduit between modules. For example, wires, cables, optical paths, waveguides and mechanical linkages can all be terminals. These terminals may represent only a slice of space separating two parts of some conductor [1].

Testbed - Any system whose primary purpose is to act as an experimental equipment for examining how an operational system reacts to changes internally or externally.

Testlist - The database that contains all of the test information for the FIS knowledge database.

Test point - This is a terminal in a module where a diagnostic test can be performed.

Upstream faults - This is a fault in a module connected to the module test point at a location that acts on the signal at some time prior to the time it passed the test location.

UUT (unit under test) - This is the hardware system or subsystem that is being diagnostically examined.

Virtual module - or pseudomodule - A module which exists as a logical construct of a causal rule in the knowledge database. This is composed solely of pseudonodes. In general, it represents interconnections between real modules that have their physical manifestation usually as the backplane in a system. In the knowledge database it must obey the format of the causal rules. It must be defined as a module. It has at least one rule which uses the pseudomodule as a cause.

Appendix A

SAMPLE RULE SET DATA FORMAT

**********start left_channels downstream left corr**********
**********start upstream

Module: upstream_sonar_units

No	Cause	Effect	Type	Precondition
1	upstream_sonar_units	a26J3_beams volts hi	s	t
2	upstream_sonar_units	a26J3_beams volts lo	s	t
3	upstream_sonar_units	a26J3_beams uniformity bad	s	t
4	upstream_sonar_units	a26J3_beams waveform bad	s	t

[Modified 22 July 1987. upstream_sonar_units: Removed Failure Rate and Replacement Cost fields from header.]
[Modified 22 July 1987. upstream_sonar_units: Removed - from space between column labels and rule 1.]

Module: a26a1FL1

No	Cause	Effect	Type	Precondition
1	a26a1a58J3 volts hi	a26a1a58J4 volts hi	s	t
2	a26a1a58J3 volts lo	a26a1a58J4 volts lo	s	t
3	a26a1a58J3 waveform bad	a26a1a58J4 waveform bad	s	t
4	a26a1FL1	a26a1a58J4 volts hi	s	t
5	a26a1FL1	a26a1a58J4 volts lo	s	t
6	a26a1FL1	a26a1a58J4 waveform bad	s	t

Module: a26a1FL2

No	Cause	Effect	Type	Precondition
1	a26a1a59J3 volts hi	a26a1a59J4 volts hi	s	t
2	a26a1a59J3 volts lo	a26a1a59J4 volts lo	s	t
3	a26a1a59J3 waveform bad	a26a1a59J4 waveform bad	s	t
4	a26a1FL2	a26a1a59J4 volts hi	s	t
5	a26a1FL2	a26a1a59J4 volts lo	s	t
6	a26a1FL2	a26a1a59J4 waveform bad	s	t

Module: a26a1FL3

No	Cause	Effect	Type	Precondition
1	a26a1a60J3 volts hi	a26a1a60J4 volts hi	s	t
2	a26a1a60J3 volts lo	a26a1a60J4 volts lo	s	t
3	a26a1a60J3 waveform bad	a26a1a60J4 waveform bad	s	t
4	a26a1FL3	a26a1a60J4 volts hi	s	t
5	a26a1FL3	a26a1a60J4 volts lo	s	t
6	a26a1FL3	a26a1a60J4 waveform bad	s	t

Module: a26a1FL4

No	Cause	Effect	Type	Precondition
1	a26a1a61J3 volts hi	a26a1a61J4 volts hi	s	t
2	a26a1a61J3 volts lo	a26a1a61J4 volts lo	s	t
3	a26a1a61J3 waveform bad	a26a1a61J4 waveform bad	s	t
4	a26a1FL4	a26a1a61J4 volts hi	s	t
5	a26a1FL4	a26a1a61J4 volts lo	s	t
6	a26a1FL4	a26a1a61J4 waveform bad	s	t

Module: a26a1a58_mod_amp_mpx

No	Cause	Effect	Type	Precondition
1	a26J3_beams volts hi	a26a1a58J1 volts hi	s	t
2	a26J3_beams volts lo	a26a1a58J1 volts lo	s	t
3	a26J3_beams uniformity bad	a26a1a58J1 volts hi	s	t
4	a26J3_beams uniformity bad	a26a1a58J1 volts lo	s	t
5	a26J3_beams waveform bad	a26a1a58J1 waveform bad	s	t
6	a26a1a58J1 volts hi	a26a1a58J3 volts hi	s	t
7	a26a1a58J1 volts lo	a26a1a58J3 volts lo	s	t
8	a26a1a58J1 waveform bad	a26a1a58J3 waveform bad	s	t
9	a26a1a70J3 reference_signal bad	a26a1a58J3 volts hi	s	t
10	a26a1a70J3 reference_signal bad	a26a1a58J3 volts lo	s	t
11	a26a1a70J3 reference_signal bad	a26a1a58J3 waveform bad	s	t
12	[Deleted 21 July 1987. Moved to a26a1FL1.]			
13	[Deleted 21 July 1987. Moved to a26a1FL1.]			
14	[Deleted 21 July 1987. Moved to a26a1FL1.]			
15	[Deleted 21 July 1987. Moved to a26a1FL1.]			
16	[Deleted 21 July 1987. Moved to a26a1FL1.]			
17	[Deleted 21 July 1987. Moved to a26a1FL1.]			
18	a26a1a58J4 volts hi	a26a1S9-1 time_slot_1_volts hi	s	t
19	a26a1a58J4 volts lo	a26a1S9-1 time_slot_1_volts lo	s	t
20	a26a1a58J4 waveform bad	a26a1S9-1 time_slot_1_waveform bad	s	t
21	a26a1a58_mod_amp_mpx	a26a1S9-1 time_slot_1_volts hi	s	t
22	a26a1a58_mod_amp_mpx	a26a1S9-1 time_slot_1_volts lo	s	t
23	a26a1a58_mod_amp_mpx	a26a1S9-1 time_slot_1_waveform bad	s	t
24	a26a1a58J7 gate_select bad	a26a1S9-1 time_slot_1_volts hi	s	t
25	a26a1a58J7 gate_select bad	a26a1S9-1 time_slot_1_volts lo	s	t
26	a26a1a58J7 gate_select bad	a26a1S9-1 time_slot_1_waveform bad	s	t

27 a26a76J3 volts bad [+12v supply]	a26a1S9-1 time_slot_1_volts hi	s	t
28 a26a76J3 volts bad	a26a1S9-1 time_slot_1_volts lo	s	t
29 a26a76J3 volts bad	a26a1S9-1 time_slot_1_waveform bad	s	t
30 a26a57-23 volts bad [-12v supply]	a26a1S9-1 time_slot_1_volts hi	s	t
31 a26a57-23 volts bad	a26a1S9-1 time_slot_1_volts lo	s	t
32 a26a57-23 volts bad	a26a1S9-1 time_slot_1_waveform bad	s	t
33 a26a77J4 volts bad [+4.5v supply]	a26a1S9-1 time_slot_1_volts hi	s	t
34 a26a77J4 volts bad	a26a1S9-1 time_slot_1_volts lo	s	t
35 a26a77J4 volts bad	a26a1S9-1 time_slot_1_waveform bad	s	t
36 a26a77J7 volts bad [-2v supply]	a26a1S9-1 time_slot_1_volts hi	s	t
37 a26a77J7 volts bad	a26a1S9-1 time_slot_1_volts lo	s	t
38 a26a77J7 volts bad	a26a1S9-1 time_slot_1_waveform bad	s	t
39 a26a75J2 volts bad [+25v supply]	a26a1S9-1 time_slot_1_volts hi	s	t
40 a26a75J2 volts bad	a26a1S9-1 time_slot_1_volts lo	s	t
41 a26a75J2 volts bad	a26a1S9-1 time_slot_1_waveform bad	s	t
42 a26a78J5 volts bad [-25v supply]	a26a1S9-1 time_slot_1_volts hi	s	t
43 a26a78J5 volts bad	a26a1S9-1 time_slot_1_volts lo	s	t
44 a26a78J5 volts bad	a26a1S9-1 time_slot_1_waveform bad	s	t

[Modified 21 July 1987. a26a1a58_mod_amp_mpx: Moved rules 12-17 to a26a1FL1.]

Module: a26a1a59_mod_amp_mpx

No	Cause	Effect	Type	Precondition
1	a26a1a59J1 volts hi	a26a1a59J3 volts hi	s	t
2	a26a1a59J1 volts lo	a26a1a59J3 volts lo	s	t
3	a26a1a59J1 waveform bad	a26a1a59J3 waveform bad	s	t
4	a26a1a70J3 reference_signal bad	a26a1a59J3 volts hi	s	t
5	a26a1a70J3 reference_signal bad	a26a1a59J3 volts lo	s	t
6	a26a1a70J3 reference_signal bad	a26a1a59J3 waveform bad	s	t
7	[Deleted 21 July 1987. Moved to a26a1FL2.]			
8	[Deleted 21 July 1987. Moved to a26a1FL2.]			
9	[Deleted 21 July 1987. Moved to a26a1FL2.]			
10	[Deleted 21 July 1987. Moved to a26a1FL2.]			
11	[Deleted 21 July 1987. Moved to a26a1FL2.]			
12	[Deleted 21 July 1987. Moved to a26a1FL2.]			
13	a26a1a59J4 volts hi	a26a1S9-1 time_slot_4_volts hi	s	t
14	a26a1a59J4 volts lo	a26a1S9-1 time_slot_4_volts lo	s	t
15	a26a1a59J4 waveform bad	a26a1S9-1 time_slot_4_waveform bad	s	t
16	a26a1a59_mod_amp_mpx	a26a1S9-1 time_slot_4_volts hi	s	t
17	a26a1a59_mod_amp_mpx	a26a1S9-1 time_slot_4_volts lo	s	t
18	a26a1a59_mod_amp_mpx	a26a1S9-1 time_slot_4_waveform bad	s	t
19	a26a1a59J7 gate_select bad	a26a1S9-1 time_slot_4_volts hi	s	t
20	a26a1a59J7 gate_select bad	a26a1S9-1 time_slot_4_volts lo	s	t
21	a26a1a59J7 gate_select bad	a26a1S9-1 time_slot_4_waveform bad	s	t
22	a26a76J3 volts bad [+12v supply]	a26a1S9-1 time_slot_4_volts hi	s	t
23	a26a76J3 volts bad	a26a1S9-1 time_slot_4_volts lo	s	t
24	a26a76J3 volts bad	a26a1S9-1 time_slot_4_waveform bad	s	t
25	a26a57-23 volts bad [-12v supply]	a26a1S9-1 time_slot_4_volts hi	s	t
26	a26a57-23 volts bad	a26a1S9-1 time_slot_4_volts lo	s	t
27	a26a57-23 volts bad	a26a1S9-1 time_slot_4_waveform bad	s	t
28	a26a77J4 volts bad [+4.5v supply]	a26a1S9-1 time_slot_4_volts hi	s	t

29 a26a77J4 volts bad	a26a1S9-1 time_slot_4_volts lo	s	t
30 a26a77J4 volts bad	a26a1S9-1 time_slot_4_waveform bad	s	t
31 a26a77J7 volts bad [-2v supply]	a26a1S9-1 time_slot_4_volts hi	s	t
32 a26a77J7 volts bad	a26a1S9-1 time_slot_4_volts lo	s	t
33 a26a77J7 volts bad	a26a1S9-1 time_slot_4_waveform bad	s	t
34 a26a75J2 volts bad [+25v supply]	a26a1S9-1 time_slot_4_volts hi	s	t
35 a26a75J2 volts bad	a26a1S9-1 time_slot_4_volts lo	s	t
36 a26a75J2 volts bad	a26a1S9-1 time_slot_4_waveform bad	s	t
37 a26a78J5 volts bad [-25v supply]	a26a1S9-1 time_slot_4_volts hi	s	t
38 a26a78J5 volts bad	a26a1S9-1 time_slot_4_volts lo	s	t
39 a26a78J5 volts bad	a26a1S9-1 time_slot_4_waveform bad	s	t

[Modified 21 July 1987. a26a1a59_mod_amp_mpx: Moved rules 7-12 to a26a1FL2.]
[Modified 21 July 1987. a26a1a59_mod_amp_mpx: Removed failure rate ane replacement cost fields from heading.]

Module: a26a1a60_mod_amp_mpx

No	Cause	Effect	Type	Precondition
1	a26J3_beams volts hi	a26a1a60J1 volts hi	s	t
2	a26J3_beams volts lo	a26a1a60J1 volts lo	s	t
3	a26J3_beams uniformity bad	a26a1a60J1 volts hi	s	t
4	a26J3_beams uniformity bad	a26a1a60J1 volts lo	s	t
5	a26J3_beams waveform bad	a26a1a60J1 waveform bad	s	t
6	a26a1a60J1 volts hi	a26a1a60J3 volts hi	s	t
7	a26a1a60J1 volts lo	a26a1a60J3 volts lo	s	t
8	a26a1a60J1 waveform bad	a26a1a60J3 waveform bad	s	t
9	a26a1a70J3 reference_signal bad	a26a1a60J3 volts hi	s	t
10	a26a1a70J3 reference_signal bad	a26a1a60J3 volts lo	s	t
11	a26a1a70J3 reference_signal bad	a26a1a60J3 waveform bad	s	t
12	[Deleted 21 July 1987. Moved to a26a1FL3.]			
13	[Deleted 21 July 1987. Moved to a26a1FL3.]			
14	[Deleted 21 July 1987. Moved to a26a1FL3.]			
15	[Deleted 21 July 1987. Moved to a26a1FL3.]			
16	[Deleted 21 July 1987. Moved to a26a1FL3.]			
17	[Deleted 21 July 1987. Moved to a26a1FL3.]			
18	a26a1a60J4 volts hi	a26a1S9-1 time_slot_7_volts hi	s	t
19	a26a1a60J4 volts lo	a26a1S9-1 time_slot_7_volts lo	s	t
20	a26a1a60J4 waveform bad	a26a1S9-1 time_slot_7_waveform bad	s	t
21	a26a1a60_mod_amp_mpx	a26a1S9-1 time_slot_7_volts hi	s	t
22	a26a1a60_mod_amp_mpx	a26a1S9-1 time_slot_7_volts lo	s	t
23	a26a1a60_mod_amp_mpx	a26a1S9-1 time_slot_7_waveform bad	s	t
24	a26a1a60J7 gate_select bad	a26a1S9-1 time_slot_7_volts hi	s	t
25	a26a1a60J7 gate_select bad	a26a1S9-1 time_slot_7_volts lo	s	t
26	a26a1a60J7 gate_select bad	a26a1S9-1 time_slot_7_waveform bad	s	t
27	a26a76J3 volts bad [+12v supply]	a26a1S9-1 time_slot_7_volts hi	s	t
28	a26a76J3 volts bad	a26a1S9-1 time_slot_7_volts lo	s	t
29	a26a76J3 volts bad	a26a1S9-1 time_slot_7_waveform bad	s	t
30	a26a57-23 volts bad [-12v supply]	a26a1S9-1 time_slot_7_volts hi	s	t
31	a26a57-23 volts bad	a26a1S9-1 time_slot_7_volts lo	s	t
32	a26a57-23 volts bad	a26a1S9-1 time_slot_7_waveform bad	s	t
33	a26a77J4 volts bad [+4.5v supply]	a26a1S9-1 time_slot_7_volts hi	s	t
34	a26a77J4 volts bad	a26a1S9-1 time_slot_7_volts lo	s	t

35 a26a77J4 volts bad	a26a1S9-1 time_slot_7_waveform bad	s	t
36 a26a77J7 volts bad [-2v supply]	a26a1S9-1 time_slot_7_volts hi	s	t
37 a26a77J7 volts bad	a26a1S9-1 time_slot_7_volts lo	s	t
38 a26a77J7 volts bad	a26a1S9-1 time_slot_7_waveform bad	s	t
39 a26a75J2 volts bad [+25v supply]	a26a1S9-1 time_slot_7_volts hi	s	t
40 a26a75J2 volts bad	a26a1S9-1 time_slot_7_volts lo	s	t
41 a26a75J2 volts bad	a26a1S9-1 time_slot_7_waveform bad	s	t
42 a26a78J5 volts bad [-25v supply]	a26a1S9-1 time_slot_7_volts hi	s	t
43 a26a78J5 volts bad	a26a1S9-1 time_slot_7_volts lo	s	t
44 a26a78J5 volts bad	a26a1S9-1 time_slot_7_waveform bad	s	t

[Modified 21 July 1987. a26a1a60_mod_amp_mpx: Moved rules 12-17 to a26a1FL3.]
[Modified 21 July 1987. a26a1a60_mod_amp_mpx: Removed failure rate ane replacement cost fields from heading.]

Module: a26a1a61_mod_amp_mpx

No	Cause	Effect	Type	Precondition
1	a26J3_beams volts hi	a26a1a61J1 volts hi	s	t
2	a26J3_beams volts lo	a26a1a61J1 volts lo	s	t
3	a26J3_beams uniformity bad	a26a1a61J1 volts hi	s	t
4	a26J3_beams uniformity bad	a26a1a61J1 volts lo	s	t
5	a26J3_beams waveform bad	a26a1a61J1 waveform bad	s	t
6	a26a1a61J1 volts hi	a26a1a61J3 volts hi	s	t
7	a26a1a61J1 volts lo	a26a1a61J3 volts lo	s	t
8	a26a1a61J1 waveform bad	a26a1a61J3 waveform bad	s	t
9	a26a1a70J3 reference_signal bad	a26a1a61J3 volts hi	s	t
10	a26a1a70J3 reference_signal bad	a26a1a61J3 volts lo	s	t
11	a26a1a70J3 reference_signal bad	a26a1a61J3 waveform bad	s	t
12	[Deleted 21 July 1987. Moved to a26a1FL4.]			
13	[Deleted 21 July 1987. Moved to a26a1FL4.]			
14	[Deleted 21 July 1987. Moved to a26a1FL4.]			
15	[Deleted 21 July 1987. Moved to a26a1FL4.]			
16	[Deleted 21 July 1987. Moved to a26a1FL4.]			
17	[Deleted 21 July 1987. Moved to a26a1FL4.]			
18	a26a1a61J4 volts hi	a26a1S9-1 time_slot_10_volts hi	s	t
19	a26a1a61J4 volts lo	a26a1S9-1 time_slot_10_volts lo	s	t
20	a26a1a61J4 waveform bad	a26a1S9-1 time_slot_10_waveform bad	s	t
21	a26a1a61_mod_amp_mpx	a26a1S9-1 time_slot_10_volts hi	s	t
22	a26a1a61_mod_amp_mpx	a26a1S9-1 time_slot_10_volts lo	s	t
23	a26a1a61_mod_amp_mpx	a26a1S9-1 time_slot_10_waveform bad	s	t
24	a26a1a61J7 gate_select bad	a26a1S9-1 time_slot_10_volts hi	s	t
25	a26a1a61J7 gate_select bad	a26a1S9-1 time_slot_10_volts lo	s	t
26	a26a1a61J7 gate_select bad	a26a1S9-1 time_slot_10_waveform bad	s	t
27	a26a76J3 volts bad [+12v supply]	a26a1S9-1 time_slot_10_volts hi	s	t
28	a26a76J3 volts bad	a26a1S9-1 time_slot_10_volts lo	s	t
29	a26a76J3 volts bad	a26a1S9-1 time_slot_10_waveform bad	s	t
30	a26a57-23 volts bad [-12v supply]	a26a1S9-1 time_slot_10_volts hi	s	t
31	a26a57-23 volts bad	a26a1S9-1 time_slot_10_volts lo	s	t
32	a26a57-23 volts bad	a26a1S9-1 time_slot_10_waveform bad	s	t
33	a26a77J4 volts bad [+4.5v supply]	a26a1S9-1 time_slot_10_volts hi	s	t
34	a26a77J4 volts bad	a26a1S9-1 time_slot_10_volts lo	s	t
35	a26a77J4 volts bad	a26a1S9-1 time_slot_10_waveform bad	s	t

36 a26a77J7 volts bad [-2v supply]	a26a1S9-1 time_slot_10_volts hi	s	t
37 a26a77J7 volts bad	a26a1S9-1 time_slot_10_volts lo	s	t
38 a26a77J7 volts bad	a26a1S9-1 time_slot_10_waveform bad	s	t
39 a26a75J2 volts bad [+25v supply]	a26a1S9-1 time_slot_10_volts hi	s	t
40 a26a75J2 volts bad	a26a1S9-1 time_slot_10_volts lo	s	t
41 a26a75J2 volts bad	a26a1S9-1 time_slot_10_waveform bad	s	t
42 a26a78J5 volts bad [-25v supply]	a26a1S9-1 time_slot_10_volts hi	s	t
43 a26a78J5 volts bad	a26a1S9-1 time_slot_10_volts lo	s	t
44 a26a78J5 volts bad	a26a1S9-1 time_slot_10_waveform bad	s	t

[Modified 21 July 1987. a26a1a61_mod_amp_mpx: Moved rules 12-17 to a26a1FL4.]
[Modified 21 July 1987. a26a1a61_mod_amp_mpx: Removed failure rate ane replacement cost fields from heading.]

**********start downstream
Module: a26a1a45_doppler_det

No Cause	Effect	Type	Precondition
1 a26a1a30J4 amplitude hi	a26a1a45J2 amplitude hi	s	t
2 a26a1a45J2 amplitude hi	a26a1a45J3 amplitude hi	s	t
3 a26a1a45J3 amplitude hi	a26a1a45J5 amplitude hi	s	t
4 a26a1a45J5 amplitude hi	a26J2-f CP_beam_1_amplitude hi	s	t
5 a26a1a30J4 amplitude lo	a26a1a45J2 amplitude lo	s	t
6 a26a1a45J2 amplitude lo	a26a1a45J3 amplitude lo	s	t
7 a26a1a45J3 amplitude lo	a26a1a45J5 amplitude lo	s	t
8 a26a1a45J5 amplitude lo	a26J2-f CP_beam_1_amplitude lo	s	t
9 a26a1a30J4 waveform bad	a26a1a45J2 waveform bad	s	t
10 a26a1a45J2 waveform bad	a26a1a45J3 waveform bad	s	t
11 a26a1a45J3 waveform bad	a26a1a45J5 waveform bad	s	t
12 a26a1a45J5 waveform bad	a26J2-f CP_beam_1_waveform bad	s	t
13 a26a1a42J5 carrier bad	a26a1a45J4 carrier bad	a	t
14 a26a1a45J4 carrier bad	a26a1a45J5 amplitude hi	s	t
15 a26a1a45J4 carrier bad	a26a1a45J5 amplitude lo	s	t
16 a26a1a45J4 carrier bad	a26a1a45J5 waveform bad	s	t
17 a26a1a45 fil_det_functions bad	a26a1a45J2 amplitude hi	s	t
18 a26a1a45_doppler_det	a26a1a45J3 amplitude hi	s	t
19 a26a1a45 mod_amp_functions bad	a26a1a45J5 amplitude hi	s	t
20 a26a1a45 mod_amp_functions bad	a26J2-f CP_beam_1_amplitude hi	s	t
21 a26a1a45 fil_det_functions bad	a26a1a45J2 amplitude lo	s	t
22 a26a1a45_doppler_det	a26a1a45J3 amplitude lo	s	t
23 a26a1a45 mod_amp_functions bad	a26a1a45J5 amplitude lo	s	t
24 a26a1a45 mod_amp_functions bad	a26J2-f CP_beam_1_amplitude lo	s	t
25 a26a1a45 fil_det_functions bad	a26a1a45J2 waveform bad	s	t
26 a26a1a45_doppler_det	a26a1a45J3 waveform bad	s	t
27 a26a1a45 mod_amp_functions bad	a26a1a45J5 waveform bad	s	t
28 a26a1a45 mod_amp_functions bad	a26J2-f CP_beam_1_waveform bad	s	t
29 a26a1a45_doppler_det	a26a1a45 fil_det_functions bad	s	t
30 a26a1a45_doppler_det	a26a1a45 mod_amp_functions bad	s	t
31 a26a1a74J1 volts bad [25v supply]	a26a1a45 fil_det_functions bad	s	t
32 a26a1a74J1 volts bad [25v supply]	a26a1a45 mod_amp_functions bad	s	t
33 a26a1a79J6 volts bad [-6v supply]	a26a1a45 mod_amp_functions bad	s	t

Module: a26a1a46_doppler_det

No	Cause	Effect	Type	Precondition
1	a26a1a31J4 amplitude hi	a26a1a46J2 amplitude hi	s	t
2	a26a1a46J2 amplitude hi	a26a1a46J3 amplitude hi	s	t
3	a26a1a46J3 amplitude hi	a26a1a46J5 amplitude hi	s	t
4	a26a1a46J5 amplitude hi	a26J2-Q CP_beam_2_amplitude hi	s	t
5	a26a1a46J5 amplitude hi	a26a1a46-22 amplitude hi	s	t
6	a26a1a31J4 amplitude lo	a26a1a46J2 amplitude lo	s	t
7	a26a1a46J2 amplitude lo	a26a1a46J3 amplitude lo	s	t
8	a26a1a46J3 amplitude lo	a26a1a46J5 amplitude lo	s	t
9	a26a1a46J5 amplitude lo	a26J2-Q CP_beam_2_amplitude lo	s	t
10	a26a1a46J5 amplitude lo	a26a1a46-22 amplitude lo	s	t
11	a26a1a31J4 waveform bad	a26a1a46J2 waveform bad	s	t
12	a26a1a46J2 waveform bad	a26a1a46J3 waveform bad	s	t
13	a26a1a46J3 waveform bad	a26a1a46J5 waveform bad	s	t
14	a26a1a46J5 waveform bad	a26J2-Q CP_beam_2_waveform bad	s	t
15	a26a1a46J5 waveform bad	a26a1a46-22 waveform bad	s	t
16	a26a1a42J5 carrier bad	a26a1a46J4 carrier bad	a	t
17	a26a1a46J4 carrier bad	a26a1a46J5 amplitude hi	s	t
18	a26a1a46J4 carrier bad	a26a1a46J5 amplitude lo	s	t
19	a26a1a46J4 carrier bad	a26a1a46J5 waveform bad	s	t
20	a26a1a46 fil_det_functions bad	a26a1a46J2 amplitude hi	s	t
21	a26a1a46_doppler_det	a26a1a46J3 amplitude hi	s	t
22	a26a1a46 mod_amp_functions bad	a26a1a46J5 amplitude hi	s	t
23	a26a1a46 mod_amp_functions bad	a26J2-Q CP_beam_2_amplitude hi	s	t
24	a26a1a46 mod_amp_functions bad	a26a1a46-22 amplitude hi	s	t
25	a26a1a46 fil_det_functions bad	a26a1a46J2 amplitude lo	s	t
26	a26a1a46_doppler_det	a26a1a46J3 amplitude lo	s	t
27	a26a1a46 mod_amp_functions bad	a26a1a46J5 amplitude lo	s	t
28	a26a1a46 mod_amp_functions bad	a26J2-Q CP_beam_2_amplitude lo	s	t
29	a26a1a46 mod_amp_functions bad	a26a1a46-22 amplitude lo	s	t
30	a26a1a46 fil_det_functions bad	a26a1a46J2 waveform bad	s	t
31	a26a1a46_doppler_det	a26a1a46J3 waveform bad	s	t
32	a26a1a46 mod_amp_functions bad	a26a1a46J5 waveform bad	s	t
33	a26a1a46 mod_amp_functions bad	a26J2-Q CP_beam_2_waveform bad	s	t
34	a26a1a46 mod_amp_functions bad	a26a1a46-22 waveform bad	s	t
35	a26a1a46_doppler_det	a26a1a46 fil_det_functions bad	s	t
36	a26a1a46_doppler_det	a26a1a46 mod_amp_functions bad	s	t
37	a26a1a74J1 volts bad [25v supply]	a26a1a46 fil_det_functions bad	s	t
38	a26a1a74J1 volts bad [25v supply]	a26a1a46 mod_amp_functions bad	s	t
39	a26a1a79J6 volts bad [-6v supply]	a26a1a46 mod_amp_functions bad	s	t

Module: a26a1a47_doppler_det

No	Cause	Effect	Type	Precondition
1	a26a1a32J4 amplitude hi	a26a1a47J2 amplitude hi	s	t
2	a26a1a47J2 amplitude hi	a26a1a47J3 amplitude hi	s	t
3	a26a1a47J3 amplitude hi	a26a1a47J5 amplitude hi	s	t
4	a26a1a47J5 amplitude hi	a26J2-K CP_beam_3_amplitude hi	s	t
5	a26a1a32J4 amplitude lo	a26a1a47J2 amplitude lo	s	t

6 a26a1a47J2 amplitude lo	a26a1a47J3 amplitude lo	s	t
7 a26a1a47J3 amplitude lo	a26a1a47J5 amplitude lo	s	t
8 a26a1a47J5 amplitude lo	a26J2-K CP_beam_3_amplitude lo	s	t
9 a26a1a32J4 waveform bad	a26a1a47J2 waveform bad	s	t
10 a26a1a47J2 waveform bad	a26a1a47J3 waveform bad	s	t
11 a26a1a47J3 waveform bad	a26a1a47J5 waveform bad	s	t
12 a26a1a47J5 waveform bad	a26J2-K CP_beam_3_waveform bad	s	t
13 a26a1a42J5 carrier bad	a26a1a47J4 carrier bad	a	t
14 a26a1a47J4 carrier bad	a26a1a47J5 amplitude hi	s	t
15 a26a1a47J4 carrier bad	a26a1a47J5 amplitude lo	s	t
16 a26a1a47J4 carrier bad	a26a1a47J5 waveform bad	s	t
17 a26a1a47 fil_det_functions bad	a26a1a47J2 amplitude hi	s	t
18 a26a1a47_doppler_det	a26a1a47J3 amplitude hi	s	t
19 a26a1a47 mod_amp_functions bad	a26a1a47J5 amplitude hi	s	t
20 a26a1a47 mod_amp_functions bad	a26J2-K CP_beam_3_amplitude hi	s	t
21 a26a1a47 fil_det_functions bad	a26a1a47J2 amplitude lo	s	t
22 a26a1a47_doppler_det	a26a1a47J3 amplitude lo	s	t
23 a26a1a47 mod_amp_functions bad	a26a1a47J5 amplitude lo	s	t
24 a26a1a47 mod_amp_functions bad	a26J2-K CP_beam_3_amplitude lo	s	t
25 a26a1a47 fil_det_functions bad	a26a1a47J2 waveform bad	s	t
26 a26a1a47_doppler_det	a26a1a47J3 waveform bad	s	t
27 a26a1a47 mod_amp_functions bad	a26a1a47J5 waveform bad	s	t
28 a26a1a47 mod_amp_functions bad	a26J2-K CP_beam_3_waveform bad	s	t
29 a26a1a47_doppler_det	a26a1a47 fil_det_functions bad	s	t
30 a26a1a47_doppler_det	a26a1a47 mod_amp_functions bad	s	t
31 a26a1a74J1 volts bad [25v supply]	a26a1a47 fil_det_functions bad	s	t
32 a26a1a74J1 volts bad [25v supply]	a26a1a47 mod_amp_functions bad	s	t
33 a26a1a79J6 volts bad [-6v supply]	a26a1a47 mod_amp_functions bad	s	t

Module: a26a1a48_doppler_det

No Cause	Effect	Type	Precondition
1 a26a1a33J4 amplitude hi	a26a1a48J2 amplitude hi	s	t
2 a26a1a48J2 amplitude hi	a26a1a48J3 amplitude hi	s	t
3 a26a1a48J3 amplitude hi	a26a1a48J5 amplitude hi	s	t
4 a26a1a48J5 amplitude hi	a26J2-S CP_beam_4_amplitude hi	s	t
5 a26a1a33J4 amplitude lo	a26a1a48J2 amplitude lo	s	t
6 a26a1a48J2 amplitude lo	a26a1a48J3 amplitude lo	s	t
7 a26a1a48J3 amplitude lo	a26a1a48J5 amplitude lo	s	t
8 a26a1a48J5 amplitude lo	a26J2-S CP_beam_4_amplitude lo	s	t
9 a26a1a33J4 waveform bad	a26a1a48J2 waveform bad	s	t
10 a26a1a48J2 waveform bad	a26a1a48J3 waveform bad	s	t
11 a26a1a48J3 waveform bad	a26a1a48J5 waveform bad	s	t
12 a26a1a48J5 waveform bad	a26J2-S CP_beam_4_waveform bad	s	t
13 a26a1a42J5 carrier bad	a26a1a48J4 carrier bad	a	t
14 a26a1a48J4 carrier bad	a26a1a48J5 amplitude hi	s	t
15 a26a1a48J4 carrier bad	a26a1a48J5 amplitude lo	s	t
16 a26a1a48J4 carrier bad	a26a1a48J5 waveform bad	s	t
17 a26a1a48 fil_det_functions bad	a26a1a48J2 amplitude hi	s	t
18 a26a1a48_doppler_det	a26a1a48J3 amplitude hi	s	t
19 a26a1a48 mod_amp_functions bad	a26a1a48J5 amplitude hi	s	t
20 a26a1a48 mod_amp_functions bad	a26J2-S CP_beam_4_amplitude hi	s	t

21 a26a1a48 fil_det_functions bad	a26a1a48J2 amplitude lo	s	t
22 a26a1a48_doppler_det	a26a1a48J3 amplitude lo	s	t
23 a26a1a48 mod_amp_functions bad	a26a1a48J5 amplitude lo	s	t
24 a26a1a48 mod_amp_functions bad	a26J2-S CP_beam_4_amplitude lo	s	t
25 a26a1a48 fil_det_functions bad	a26a1a48J2 waveform bad	s	t
26 a26a1a48_doppler_det	a26a1a48J3 waveform bad	s	t
27 a26a1a48 mod_amp_functions bad	a26a1a48J5 waveform bad	s	t
28 a26a1a48 mod_amp_functions bad	a26J2-S CP_beam_4_waveform bad	s	t
29 a26a1a48_doppler_det	a26a1a48 fil_det_functions bad	s	t
30 a26a1a48_doppler_det	a26a1a48 mod_amp_functions bad	s	t
31 a26a1a74J1 volts bad [25v supply]	a26a1a48 fil_det_functions bad	s	t
32 a26a1a74J1 volts bad [25v supply]	a26a1a48 mod_amp_functions bad	s	t
33 a26a1a79J6 volts bad [-6v supply]	a26a1a48 mod_amp_functions bad	s	t

Module: a26a1a80_relay_board

No Cause	Effect	Type	Precondition
1 a26J3_beams volts hi	a26a1a59J1 volts hi	s	t
5 a26J3_beams volts lo	a26a1a59J1 volts lo	s	t
9 a26J3_beams uniformity bad	a26a1a59J1 volts hi	s	t
13 a26J3_beams uniformity bad	a26a1a59J1 volts lo	s	t
17 a26J3_beams waveform bad	a26a1a59J1 waveform bad	s	t

[Note 1: Relay is active when it is not supposed to be. a26a1a63J1 and a26a1a65J1 are open circuited, but a]
[signal is expected.]
[Note 2: Relay is inactive when it is supposed to be active. We expect a26a1a63J1 and a26a1a65J1 to be open]
[circuited, but they have signals. Signal level is normal for not_BB/TRK_mode, but is too high for BB/TRK_mode]

[Modified 23 June 1987. a26a1a80_relay_board:]
[Modified 23 June 1987. a26a1a80_relay_board: Changed preconditions, rules 31-33 and 40-42, from]
[BB/TRK_mode to t.]
[Modified 23 June 1987. a26a1a80_relay_board: Changed preconditions, rules 34-39. from not_BB/TRK_mode to t.]
[note that this was appended to rules.all in the jul2287 directory and that this module is the only one that has]
[[been changed]
[this was performed on august 21, 1987]
[modified september 21, 1987. a26a1a80_relay_board: changed lo to bad, cause of rules 29 and 30.]
[modified september 21, 1987. a26a1a80_relay_board: removed spurious character from "26a1a80" incause of rules]
[21,22,25 and a26.]

Module: downstream_sonar_units

No Cause	Effect	Type	Precondition
1 a26J2-f CP_beam_1_amplitude hi	display CP_beam_1_amplitude hi	s	t
2 a26J2-f CP_beam_1_amplitude lo	display CP_beam_1_amplitude lo	s	t
3 a26J2-f CP_beam_1_waveform bad	display CP_beam_1_waveform bad	s	t
4 a26J2-Q CP_beam_2_amplitude hi	display CP_beam_2_amplitude hi	s	t
5 a26J2-Q CP_beam_2_amplitude lo	display CP_beam_2_amplitude lo	s	t
6 a26J2-Q CP_beam_2_waveform bad	display CP_beam_2_waveform bad	s	t
7 a26J2-K CP_beam_3_amplitude hi	display CP_beam_3_amplitude hi	s	t
8 a26J2-K CP_beam_3_amplitude lo	display CP_beam_3_amplitude lo	s	t

9 a26J2-K CP_beam_3_waveform bad	display CP_beam_3_waveform bad	s	t
10 a26J2-S CP_beam_4_amplitude hi	display CP_beam_4_amplitude hi	s	t
11 a26J2-S CP_beam_4_amplitude lo	display CP_beam_4_amplitude lo	s	t
12 a26J2-S CP_beam_4_waveform bad	display CP_beam_4_waveform bad	s	t
37 downstream_sonar_units	display CP_beam_1_amplitude hi	s	t
38 downstream_sonar_units	display CP_beam_1_amplitude lo	s	t
39 downstream_sonar_units	display CP_beam_1_waveform bad	s	t
40 downstream_sonar_units	display CP_beam_2_amplitude hi	s	t
41 downstream_sonar_units	display CP_beam_2_amplitude lo	s	t
42 downstream_sonar_units	display CP_beam_2_waveform bad	s	t
43 downstream_sonar_units	display CP_beam_3_amplitude hi	s	t
44 downstream_sonar_units	display CP_beam_3_amplitude lo	s	t
45 downstream_sonar_units	display CP_beam_3_waveform bad	s	t
46 downstream_sonar_units	display CP_beam_4_amplitude hi	s	t
47 downstream_sonar_units	display CP_beam_4_amplitude lo	s	t
48 downstream_sonar_units	display CP_beam_4_waveform bad	s	t

Appendix B

SAMPLE TESTLIST DATA FORMAT

NAME	$1 TEST POINT	$2 PARAMETER	UNITS	$3 POSS QUAL RES	$4 MIN	$5 MAX	COST, SECONDS	PREREQUISITES	INSTR NAME	TEXT PARAMETERS
A1A1J4	a26a1A1J4	logic_levels	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J4 active_mode on	logic_1	
A1A1J5	a26a1A1J5	mts_gate	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J5	A1J1	$6=positive
A1A1J1	a26a1A1J1	not_mts_gate	logic	bad ok			20 scope ready	A1J1 a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J1		$6 ative
A1A1J3	a26a1A1J3	end_clear	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J3	A1J3	$6=TEST-END $7=26A1S8
A1A1J7	a26a1A1J7	not_CL2	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J7	A4J6	
A1A11J6	a26a1A11J6	not_ST2	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A11J6	A11J5	
A1A1J6	a26a1A1J6	ST3	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J6	A1J6	

A1A1J8	a26a1A1J8	DST3	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J8	A1J6	
a1a77j4	a26a1a77j4	volts	volts	bad ok	4.4	4.6	10	unit-26_door open a26a1_drawer open probe_on a26a1a77j4 ac_diff_mtr ready	power	$6=ac voltmeter $7=rms
a1a76j3	a26a1a76j3	volts	volts	bad ok	11.88	12.12	10	unit-26_door open a26a1_drawer open probe_on a26a1a76j3 ac_diff_mtr ready	power	$6=ac voltmeter $7=rms
A1A1J4	a26a1A1J4	logic_levels	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J4 active_mode on	logic_1	
A1A1J9	a26a1A1J9	logic_levels	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A1J9 active_mode on	logic_1	
A1A10J9	a26a1A10J9	RE1	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A10J9 active_mode on	logic_1	
A1A10J1	a26a1A10J1	circulate	logic	bad ok			20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A10J1	A10J1	

A1A10J5	a26a1A10J5	load_end_advance	logic	bad ok		20	scope ready A10J5 a26a1_drawer open unit-26_door open [1] probe_on a26a1A10J5 active_mode on		
A1A10J4	a26a1A10J4	load_end_ref	logic	bad ok		20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A10J4 active_mode on	logic_1	
A1A10J	a26a1A10J8	DST1	logic	bad ok		20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A10J8	A1J6	
A1A10J6	a26a1A10J6	ST1	logic	bad ok		20	scope ready a26a1_drawer open unit-26_door open [1]	A1J6	
A1A10J7	a26a1A10J7	not_CL1	logic	bad ok		20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A10J7	A4J6	
A1A10J3	a26a1A10J3	end_clear	logic	bad ok		20	scope ready a26a1_drawer open unit-26_door open [1] probe_on a26a1A10J3	A1J3	$6=TEST-END $7=26A1AS8
A1A11J5	a26a1A11J5	not_ST1	logic	bad ok		20	scope ready a26a1_drawer open unit-26_door open [1]	A11J5	

Appendix C

SAMPLE PRECONDITION DATA FORMAT

```
((IFCOMBO
  (A26A1A8J0 A26A1A8J2
        A26A1A8J4
        A26A1A8J6
        A26A1A8J8
        A26A1A8J10
       A26A1A8J12
        A26A1A8J14
        A26A1A8J16
        A26A1A8J18
        A26A1A8J20
        A26A1A8J45
        A26A1A8J24
        A26A1A8J26
        A26A1A8J28
        A26A1A8J28-1
        A26A1A8J30
        A26A1A8J30-1))
 (TDATTNON (A26A1A57J8))
 (VFOSEL
  (A26A1A5J36 A26A1A5J40
        A26A1A5J42
        A26A1A5J44
        A26A1A5J46
        A26A1A5J48
        A26A1A5J50
        A26A1A5J52
        A26A1A5J54
        A26A1A5J56
        A26A1A5J58
        A26A1A5J60
        A26A1A5J62
        A26A1A5J64
        A26A1A5J66
        A26A1A5J68
        A26A1A5J70
        A26A1A5J72
        A26A1A5J74
        A26A1A7J36
        A26A1A7J38
        A26A1A8J32
        A26A1A13-1
        A26A1A13J2
        A26A1A13J40
        A26A1A13J44
        A26A1A69J40-1))
```

```
(BLANK
  (A26A1A7-1 A26A1A7J6
        A26A1A7J8
        A26A1A7J10
        A26A1A7J12
        A26A1A7J14
        A26A1A7J16
        A26A1A7J18
        A26A1A7J20
        A26A1A7J22
        A26A1A7J24
        A26A1A7J26
        A26A1A7J28
        A26A1A7J30
        A26A1A7J32
        A26A1A7J34
        A26A1A7J36
        A26A1A7J38
        A26A1A1300
        A26A1A13J2
        A26A1A14J8))
(NOTBLANK (NOT BLANK))
(TDATTNOFF (NOT TDATTNON))
(REFSEL (NOT VFOSEL)))
```

Appendix D

SAMPLE ORDER DATA FORMAT

```
((A26A1A70J4 A26A1A70J6)
 (A26A1A70J6
  (A26A1A70J8 A26A1A7J10
       A26A1A7J12
       A26A1A7J14
       A26A1A7J16
       A26A1A7J18
       A26A1A7J20
       A26A1A7J22
       A26A1A7J24
       A26A1A7J26
       A26A1A7J28
       A26A1A7J30))
 (A26A1A80J4 A26A1A80J6)
 (A26A1A80J6 A26A1A8J10)
 (A26A1A8J10 A26A1A8J12)
 (A26A1A8J12 A26A1A8J14)
 (A26A1A8J14 A26A1A8J16)
 (A26A1A8J16 A26A1A8J18)
 (A26A1A8J18 A26A1A8J20)
 (A26A1A8J24 A26A1A8J26)
 (A26A1A10J2 A26A1A10J42)
 (A26A1A10J4 A26A1A10J42)
 (A26A1A10J6 A26A1A10J42)
 (A26A1A10J8 A26A1A10J42)
 (A26A1A10J10 A26A1A10J42)
 (A26A1A10J12 A26A1A10J42)
 (A26A1A10J14 A26A1A10J42)
 (A26A1A10J16 A26A1A10J42)
 (A26A1A10J18 A26A1A10J42)
 (A26A1A10J20 A26A1A10J42)
 (A26A1A10J22 A26A1A10J42)
 (A26A1A10J24 A26A1A10J42)
 (A26A1A10J26 A26A1A10J42)
 (A26A1A10J28 A26A1A10J42)
 (A26A1A10J30 A26A1A10J42)
 (A26A1A10J32 A26A1A10J42)
 (A26A1A10J34 A26A1A10J42)
 (A26A1A10J36 A26A1A10J42)
 (A26A1A10J38 A26A1A10J42)
 (A26A1A10J40 A26A1A10J42)
 (A26A1A12J8 A26A1A12J32)
 (A26A1A12J10 A26A1A12J32)
 (A26A1A12J12 A26A1A12J32)
 (A26A1A12J14 A26A1A12J32)
 (A26A1A12J16 A26A1A12J32)
```

(A26A1A12J18 A26A1A12J32)
(A26A1A12J20 A26A1A12J32)
(A26A1A12J22 A26A1A12J32)
(A26A1A12J24 A26A1A12J32)
(A26A1A12J26 A26A1A12J32)
(A26A1A12J28 A26A1A12J32)
(A26A1A12J30 A26A1A12J32)
(A26A1A13J4 A26A1A13J6)
(A26A1A13J4 A26A1A13J8)
(A26A1A13J10 A26A1A13J12)
(A26A1A13J10 A26A1A13J14)
(A26A1A6J19 A26A1A57J10A)
(A26A1A6J20 A26A1A57J20A)
(A26A1A6J22 A26A1A57J30A)
(A26A1A6J24 A26A1A57J40A)
(A26A1A6J28 A26A1A57J50A)
(A26A1A6J30 A26A1A57J60A)
(A26A1A6J32 A26A1A57J70A)
(A26A1A6J34 A26A1A57J80A)
(A26A1A8J4 A26A1A84J50))

Appendix E

SAMPLE INSTRUCTION DATA FORMAT

Instruction Name: A30J4_w

Instruction Text:

"Connect 10X scope probe to ",$1,".

Use positive external sync from 26A1A11J18.

Set TARGET CONTROL - INITIAL BEARING control 34A3B1
to ",$7," relative.

Observe analog pulses.

Verify the following:

Pulse repetition rate = 1416 microseconds.
Pulse width = 354 microseconds.
Pulse starts 708 microseconds after sync.

{",$3,"}"

Instruction Name: A45J3_a

Instruction Text:

"Connect oscilloscope to ",$1,".

Use direct coupling.

Observe CP target pulses superimposed on approximately
150 mv of noise.

Estimate the average 0 to peak amplitude of the CP target
pu s.

Correct range is 530 to 710 millivolts, zero to peak.

{",$3,"}"

Instruction Name: A45J3_w

Instruction Text:

"Connect oscilloscope to ",$1,".

Use direct coupling.

Observe CP target pulses superimposed on approximately
150 mv of noise.

Verify the following:

CP target pulse width is approximately 30 milliseconds,
to the 50% points.
CP target pulse peaks are approximately 4 time noise
level.

{",$3,"}"

Instruction Name: A45J5_a

Instruction Text:

"Connect oscilloscope to ",$1,".

Observe modulated pulse with duration of approximately
50 milliseconds.

Measure the peak to peak amplitude of the pulse.

The correct range is ",$4," to ",$5".

{",$3,"}"

Instruction Name: A45J5_w

Instruction Text:

"Connect oscilloscope to ",$1,".

Observe modulated pulse with amplitude of approximately
21 volts peak to peak.

Verify that the average pulse duration is approximately
50 milliseconds.

{",$3,"}"

Instruction Name: A4J6

Instruction Text:

"Connect X10 scope probe to ", $1,".

Use positive internal sync.

Observe square wave.

Verify the following:

Period = approx 1.0 microseconds.
Pulse Width = approx 0.5 microsecond.
Logic Level One = +2.0 to +4.5 volts.
Logic Level Zero = +0.0 to +0.6 volt.

{",$3,"}"

Appendix F

AUTOMATIC CONVERSION PROGRAM FOR RULE AND TESTLIST DATABASES

```
/*
  This program is made to read in uut data from the format made by a generic text editor and convert to a FIS ".v"
     file.
*/

/*declarations:*/

#include <stdio.h>;
#include <sys/file.h>;
#include "string.h"
#define LINE_LENGTH    145

struct lines
{
 char info[LINE_LENGTH];
} data[10000];
char response[30], answer, answer1[5], *temp, module[30], cause[50],
    effect[50], precondition[30], name[30], test_point[30], parameter[30],
    units[20], okreadin[10], okreadin2[10], okreadin3[10],
     min[10], max[10], lilnum[10];
int  ch, ch1, i, ii, filelength, counter2, length, space, numone,
    column[LINE_LENGTH], col, storenum, nextone, place;
FILE *fopen(), *fp1, *fp2, *fp3;

main()
{
                              /*open input file*/
 do
 {
  printf("What is the name of your rule file?  ");
  scanf("%s", response);
  if ((fp1=fopen(response, "r")) == NULL)
    printf("File doesn't exist.\n");
 } while (fp1 == NULL);
 do
 {
  printf("What is the name of your test file?  ");
  scanf("%s", response);
  if ((fp3=fopen(response, "r")) == NULL)
    printf("File doesn't exist.\n");
 } while (fp3 == NULL);
```

```
                        /*open output file*/
do
{
 printf("What is the name of the output file? ");
 scanf("%s", response);
 if (ch=access(response, 0) == 0)
  {
   printf("The file already exists.  Overwrite Y/N? ");
   scanf("%s", answer1);
  }
 if ((ch==1 && (answer1[0]=='y' || answer1[0]=='Y')) || ch != 1)
   fp2=fopen(response, "w");
} while (ch == 1 && (answer1[0]=='N' || answer1[0]=='n'));                    /*write & read*/
filelength=0;
numone=0;
while (answer=fgets(data[filelength].info, LINE_LENGTH, fp1) != NULL)
  ++filelength;
fprintf(fp2, "NIL\nNIL\nNIL\n(");
i=0;                    /*filelength is how long file is*/
do
{
 if ((temp=strpbrk(data[i].info, "W")) != NULL)
   if (strncmp (temp, "WORKING", 7) == 0)  /*skip 2 lines if W is found*/
     i=i+2;
 if (((temp=strpbrk(data[i].info, "M")) != NULL) &&
            (strncmp (temp, "Modu", 4) ==0))   /*look for the letter M*/
  {
     if (numone != 0)
      {
       numone=0;                              /*print control*/
       fprintf(fp2," )))\n");
      }
     for (counter2=0; counter2<=30; ++counter2)
       module[counter2]='\0';
      for (ii=8; temp[ii] != '\n'; ++ ii)
       module[ii-8]=temp[ii];
      i=i+3;
     fprintf(fp2, " ((NAME %s) (FRATE 1)\n     (CAUSAL-RULES\n", module);
 }
  else
 {
   ii=0;
   place=3;
   if ((length=strlen(data[i].info)) > 15)
   if (((strpbrk(data[i].info, "[")==NULL) && (data[i].info[ii]!='N')) ||
      (((temp=strpbrk(data[i].info, "["))!=NULL) && (strlen(data[i].info)-strlen(temp)>5)))
   {                          /*is the line [deleted] ? */
     ii=85;
    for (counter2=85; data[i].info[counter2] != '\n'; ++counter2)
      if (data[i].info[counter2] != ' ')
       precondition[counter2-ii]=data[i].info[counter2];
        else
          ++ii;
    space=0;
```

```
    for (counter2=3; counter2<=length && space != 5; ++counter2)
     if ((cause[counter2-3]=data[i].info[counter2]) == ' ')
        ++space;
     space=0;
    for (counter2=40; counter2<=length && space != 4; ++counter2)
     if ((effect[counter2-40]=data[i].info[counter2]) == ' ')
        ++space;
     if (numone == 0)
     {
       ++numone;
       fprintf(fp2, "        (\n");
     }
        /*find if the cause is the same as the module*/
      if (strlen(strpbrk(cause, " ")) < 5)
        fprintf(fp2,"        (%s %s\n              (%s))\n",precondition,
                                cause, effect);
       else
        fprintf(fp2,"        (%s (%s)\n              (%s))\n",precondition,
                                cause, effect);
   }
  }
    /*clean up the variables*/
  for (counter2=0; counter2<=50; ++counter2)
  {
   cause[counter2]='\0';
   effect[counter2]='\0';
  }
  for (counter2=0; counter2<=30; ++counter2)
   precondition[counter2]='\0';
   ++i;
} while (i<=filelength);
fprintf(fp2," )))\n)\n");
                        /*rules database*/
filelength=0;
while (answer=fgets(data[filelength].info, LINE_LENGTH, fp3) != NULL)
  ++filelength;
fprintf(fp2, "( ");
i=0;
ch=4;
if (ch<filelength)
find_col(data[ch].info, column); /*sets up the columns into int array column*/
do
{
 if ((strpbrk(data[i].info, "*")) != NULL) /*skip over any line with **'s */
    ++i;
  else
  {
    ii=0;
   answer=data[i].info[ii];
   if (((length=strlen(data[i].info)) > 55) && (answer != ' '))
   {       /*does it begin with a character?*/
     col=1;  /*this is the column number*/
     storenum= -1;
```

```
for (counter2=0; counter2<=length && col < 6; counter2++)
 if (data[i].info[counter2] != ' ' ||
    (data[i].info[counter2] == ' ' && column[counter2] != 0))
    switch(col)
    {       /*set up variables depending on column*/
     case 1:  if (storenum == -1) storenum=counter2;
              name[counter2-storenum]=data[i].info[counter2];
               nextone=1;
               break;
     case 2:  if (storenum == -1) storenum=counter2;
              test_point[counter2-storenum]=data[i].info[counter2];
               nextone=1;
               break;
     case 3:  if (storenum == -1) storenum=counter2;
             parameter[counter2-storenum]=data[i].info[counter2];
               nextone=1;
               break;
     case 4:  if (storenum == -1) storenum=counter2;
              units[counter2-storenum]=data[i].info[counter2];
               nextone=1;
               break;
     case 5:  if (storenum == -1) storenum=counter2;
             okreadin[counter2-storenum]=data[i].info[counter2];
               nextone=1;
               break;
    }
    else
    if (nextone == 1)
    {
     ++col; /*increment the column number*/
     nextone=0; /*skip over spaces between columns without
                  incrementing col */
     storenum= -1; /*the starting place for indivdual variables*/
    }


/*************************************************************************
*       hi lo ok value (ok (min max) lo (-inf min) hi (max inf))
*       hi lo ok noval (ok lo hi)
*       ok bad   value (ok (min max) bad (-inf min) (max inf))
*       ok bad   noval (ok bad)
*************************************************************************/

/*the following method is used instead of the above column method because sometimes the data doesn't exist and the
program looks for nothing or misinterprets data found*/

    ch=63;
    if (data[i].info[ch] != ' ') /*there's s.t. there!!!*/
    {
     for (counter2=63; data[i].info[counter2] != ' '; ++counter2)
      min[counter2-63]=data[i].info[counter2];
      if (strlen (min) < 6)
       for (counter2=69; data[i].info[counter2] != ' ' ||      counter2<70; ++counter2)
        max[counter2-69]=data[i].info[counter2];
```

```
  else
  {
   ch1=67;
  if (data[i+1].info[ch1] != '\0')
  for (counter2=67; data[i+1].info[counter2] != ' ' ||
                  counter2<70; ++counter2);
  }
   max[counter2-67]=data[i+1].info[counter2];
  /*the data[i+1] is used above because if the qual val is to long,
    there's no space for the second on the same line, and it's
    located underneath*/
}
 else
 if (data[i].info[ch+1] != ' ')
  {
  for (counter2=64; data[i].info[counter2] != ' '; ++counter2)
   min[counter2-64]=data[i].info[counter2];
   if (strlen (min) < 6)
    for (counter2=70; data[i].info[counter2] != ' ' ||       counter2<71; ++counter2)
     max[counter2-70]=data[i].info[counter2];
    else
    {
     ch1=68;
    if (data[i+1].info[ch1] != '\0')
    for (counter2=68; data[i+1].info[counter2] != ' ' ||
                  counter2<71; ++counter2)
     max[counter2-68]=data[i+1].info[counter2];
    }
  }
if (atof(min) > atof(max))
  printf("Problem with min max values in %s %s.\n", name, test_point);
        /*if the min is greater than the max, there's a problem.*/
ii=58; /*this is done because sometimes the data's in the wrong space*/
for (counter2=58; counter2<60; ++counter2)
 if (data[i+1].info[counter2] != ' ')
  okreadin2[counter2-ii]=data[i+1].info[counter2];
   else
    ++ii;
if (strlen(data[i+2].info) >58)
 for (counter2=58; data[i+2].info[counter2] != ' ' &&
                counter2<strlen(data[i+2].info); ++counter2)
  okreadin3[counter2-58]=data[i+2].info[counter2];
 ii=77;
for (counter2=77; counter2<80; ++counter2)
 if (data[i].info[counter2] != ' ')
  lilnum[counter2-ii]=data[i].info[counter2];
  else
   ++ii;
```

```
/******** printing time *********/
      fprintf(fp2, "   (%s (%s %s\n          S1\n",
              name, test_point, parameter);
     if ((okreadin[0] == 'f') && (okreadin2[0] == 'o'))
       fprintf(fp2, "          ((ok) (faulted))\n");
     if ((okreadin[0] == 'a') && (okreadin2[0] == 'o'))
       fprintf(fp2, "          ((absent) (ok))\n");
     if ((okreadin[0] == 'p') && (okreadin2[0] == 'o'))
       fprintf(fp2, "          ((ok) (present))\n");
     if ((okreadin[0] == 'o') && (okreadin2[0] == 'o'))
       fprintf(fp2, "          ((ok) (on))\n");
     if ((okreadin[0] == 'b') && (okreadin2[0] == 'o'))
       if (min[0] == '\0')
        fprintf(fp2, "          ((ok) (bad))\n");
        else
        fprintf(fp2,"          ((ok (%s %s)) (bad (-inf %s) (%s inf)))\n", min, max, min, max);
     if ((okreadin[0] == 'h') && (okreadin2[0] == 'l'))
       if (min[0] == '\0')
        fprintf(fp2, "          ((ok) (lo) (hi))\n");
        else
        fprintf(fp2,
          "          ((ok ((%s %s))) (lo ((-inf %s))) (hi ((%s inf))))\n", min, max, min, max);

      fprintf(fp2, "          %s\n          D\n", units);
      fprintf(fp2, "          %s\n          NIL))\n", lilnum);
    }
   }
  for (counter2=0; counter2<=30; ++counter2)
   {
    name[counter2]='\0';
    test_point[counter2]='\0'; /*cleaning up the variables*/
    parameter[counter2]='\0';
   }
  for (counter2=0; counter2<=20; ++counter2)
    units[counter2]='\0';
  for (counter2=0; counter2<=10; ++counter2)
   {
    okreadin[counter2]='\0';
    okreadin2[counter2]='\0';
    okreadin3[counter2]='\0';
    lilnum[counter2]='\0';
    min[counter2]='\0';
    max[counter2]='\0';
   }
   ++i;
 } while (i<=filelength);
 fprintf(fp2,")\n");
 fprintf(fp2,"NIL\nNIL\nNIL\n");
 close(fp1);
 close(fp3);
 close(fp2);
}
```

```
/************************** FUNCTIONS ****************************/

find_col (string, columns)

char *string;
int columns[];

{
 int k=0, num=0;

 while (string[k] == ' ') columns[k++]=1;
 if (k==0) columns[k++]=1;
 while (string[k] != '\0')
 {
  if (string[k]==' ') columns[k]=0;
    else
     if (string[k-1] != ' ') columns[k] = num;
      else
       columns[k] = ++num;
   k++;
 }
 while (k < LINE_LENGTH) columns[k++]=0;
 return;
}

/*********************************************************************
* This function looks at a header and develops column fields:
*
*   Header1        H2   Head3          Header4
* 11111111110002200333300000004444444000000
*
* as such, to find out what column a specific piece of data
* is in..
*********************************************************************/
```

Appendix G

CONVERSION PROGRAM RULE DATABASE

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                                       *
 * Program CONVERT                                                                       *
 *                                                                                       *
 * This program converts causal rule data given in tabular form to data in a             *
 * LISP format that may be used as input to the Fault Isolation System (FIS)             *
 * package. The tabular input file consists of cause-effect relationships                *
 * for a given module, and any number of modules may be specified. CONVERT*
 * allows much flexibility with regard to the format of the input file, but              *
 * several restrictions are necessary:                                                   *
 *                                                                                       *
 * 1. Each set of cause-effect relationships for a module must be                        *
 *    seperated by a line:                                                               *
 *        Module: <module_name>                                                          *
 *    Note any number of spaces and tabs is allowed before, within, and after            *
 *    the line.                                                                          *
 * 2. Each cause-effect relationship for a module must be numbered and                   *
 *    given in the following form:                                                       *
 *        <no.> <cause> <effect> <type> <precondition>                                   *
 *    <effect> is a triplet of the following form:                                       *
 *        <terminal> <parameter> <abnormality>                                           *
 *    <cause> is either a triplet as above or an atom of the form:                       *
 *        <module_name>                                                                  *
 *    Again, any number of spaces and tabs is allowed within the line: each              *
 *    parameter field is determined by its position on the line relative to              *
 *    the others, not by its columnar location.                                          *
 * 3. The <cause> field given above may also be a conjunction of causes. If              *
 *    so, the cause-effect relationship may appear on one or more lines:                 *
 *        <no.> <cause1> & <cause2> & ... & <causen> <effect> <type> <pre. *
 *    or:                                                                                *
 *        <no.> <cause1> & ... & <causem> &                                              *
 *        <causem+1> & ... & <causek> &                                                  *
 *                 :                                                                     *
 *        <causei> & ... & <causen>   <effect> <type> <precondition>                     *
 *    Note that each cause must be seperated by '&' and that, if the list of             *
 *    causes continues to the next line, the last character on the line (other           *
 *    than a space or tab) must be '&'. Each cause may be either an atom or a            *
 *    triplet as describe above.                                                         *
 * 4. Parentheses may not appear anywhere in the input file.                             *
 * 5. Comments may appear anywhere in the file. They are delimited on the                *
 *    left by either [ or { and on the right by ] or }. Mixing of delimiters             *
 *    is not allowed: comments started with a bracket must end with a                    *
 *    bracket, and those started with a brace must end with a brace. Both                *
 *    delimiters must occur on the same line.                                            *
 * 6. The maximum number of characters allowed on an input line (including               *
 *    spaces and tabs) is MAX_LENGTH (specified in the C program).                       *
 *                                                                                       *
```

```
* Any line appearing in the file that does not follow the format described      *
* above is simply ignored. Thus headers, page numbers, spaces, etc. are        *
* permitted anywhere and will have no effect (as long as they cannot be        *
* interpreted in one of the ways described above).                             *
*                                                                              *
* The output of the program is a series of lists in which the head is the      *
* module name and the tail is a list of cause-effect relationships:            *
*                                                                              *
*        (module_name                                                          *
*         (                                                                    *
*         (cause effect [precondition])                                        *
*                 :                                                            *
*         (cause effect [precondition])                                        *
*         ))                                                                   *
*        (module_name                                                          *
*              :                                                               *
*         ))                                                                   *
*                                                                              *
* 'effect' is always of the form '(terminal parameter abnormality)' and        *
* 'cause' is either in this form or a non-parenthesised atom: 'module_name'     *
* The precondition is printed only when it is given as something other than    *
* "t". Thus for the input line                                                 *
*    4 term1 par1 effect1    term2 par2 effect2    s    t                      *
* the output will be                                                           *
*    ((term1 par1 effect1) (term2 par2 effect2))                               *
* And for the input line                                                       *
*    5 module    term3 par3 effect3    s    precond1                          *
* the corresponding output line will be                                        *
*    (module (term3 par3 effect3) precond1)                                    *
* For the case when 'cause' is a conjunction of causes, the ouput will look    *
* as follows:                                                                  *
*    ; (cause1 & cause2 & ... & causen effect type [precondition])             *
* Each cause is either a triplet (in parentheses) or an atom. Note that this   *
* line is commented out (a semicolon in LISP indicates a comment)              *
* because the the FIS package is not yet able to handle this case.             *
*                                                                              *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

#include <stdio.h>
#define MAX_LENGTH 200      /* maximum length of input line */
#define TRUE 1
#define FALSE 0
#define MAX_MODULE_NUM 200

FILE *fp1, *fp2[MAX_MODULE_NUM];  /* pointers to input, output files */


main()
{
  char string[MAX_LENGTH];    /* array into which input line is placed */
  char c[6];
  char name1[30], name2[30];  /* file names of input, output files    */
   int num, k, k1, a, i;

  do {
    printf("Enter the name of the file to be converted... ");
     scanf("%s",name1);
```

```
    if ((fp1=fopen(name1,"r")) == NULL)
       printf("No such file exists.\n");
  } while (fp1 == NULL);

  /*
    Read in lines until the first module identification is found.
  */


  num = gets_1(string);
  while (strncmp("Module:",&string[num],7) != 0)
    num = gets_1(string);
  /* Disregard spaces or tabs before module name.  */
  for (k=num+7; string[k]==' ' || string[k]=='\t'; k++) ;
  /* Place a NUL character at end of module name.  */
  for (k1=k; string[k1]!=' ' && string[k1]!='\t' && string[k1]!='\0'; k1++) ;
  string[k1] = '\0';
  i = 0;
  strcpy (name2, &string[k]);
  fp2[i] = fopen(name2,"w+");

  /*
     This is the main loop in the program.  Each line is read until EOF is
    encountered.  If the first character in the line (other than a space or
    tab) is a digit, then routine 'sep' is called to print the output line if
    it is in the proper format.  If the first character is not a digit, then
    a check is made to see if the line is a module identification.  If so, the
    module name is printed, otherwise no output is given for this line. The
    routine 'strncmp(s1,s2,n)' compares the first n characters of strings s1
    and s2 and returns 0 if they are the same.
  */

  while((num=gets_1(string)) != 999)
    if (string[num]>='0' && string[num]<='9')  /* digit check */
       sep(&string[num],i);
    else if (strncmp("Module:",&string[num],7) == 0) {  /* New module found */
       fclose (fp2[i++]); /*close the last module file */
       for (k=num+7; string[k]==' ' || string[k]=='\t'; k++) ;
       for (k1=k; string[k1]!=' ' && string[k1]!='\t' && string[k1]!='\0';
            k1++) ;
       string[k1] = '\0';  /* Place NUL character at end of module name. */
       strcpy (name2, &string[k]);
       fp2[i] = fopen(name2,"w+"); /* open the new module file */
    }

  /* Close input and output files.  */
  fclose(fp1);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                           *
 * SEP(STRING,I)                                                             *
 *                                                                           *
 * Routine SEP receives an input line beginning at location STRING.  This    *
 * input will be recognized (i.e. output wil be generated) if given in the   *
 * form descibed previously, that is:                                        *
 *      <no.> <cause> <effect> <type> <precondition>                         *
```

```
* where <effect> is a triplet and <cause> is either a triplet or an atom and    *
 * may be a conjunction of several causes.                                      *
 * SEP performs character by character examination of the input string,         *
 * during which it is always in one of four possible states (or MODEs):         *
 *   MODE = 0:  An input parameter is being read.                               *
 *   MODE = 1:  Spaces or tabs are being read.                                  *
 *   MODE = 2:  A comment is being read (bracket delimeters).                   *
 *   MODE = 3:  A comment is being read (brace delimeters).                     *
 * The array of pointers DATA stores the locations of the parameters on the     *
 * input line. After the string has been processed, a check is made to see if   *
 * the input is in the correct format. If so, the appropriate output line is    *
 * printed.                                                                     *
 *                                                                              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

sep(string,i)
char *string;
int i;
{
  int mode = 1, m = 0, n1 = -1;
  int n, k, num, k1, k2, OK, LOOP;
  char *loc;
  char *strrchr();
  char data[50][50];       /* stores a max of 50 parameter fields */
  char s1[MAX_LENGTH];     /* array in which an input line following a
                              conjunction of causes is placed        */


  do {
    n = -1;
    while (string[++n] != 0)
      if (mode == 0) {
         if (string[n] != ' ' && string[n] != '\t')
           data[n1][m++] = string[n]; /* update current parameter field */
         else {   /* end of current parameter field */
            mode = 1;
           data[n1][m] = 0;  /* place NULL character at end of string */
          }
       }
      else if (mode == 1) {
         if (string[n] == '[') mode = 2;  /* start of a comment  */
         else if (string[n] == '{') mode = 3; /* start of a comment  */
         else if ((string[n] != ' ') && (string[n] != '\t')) {
           mode = 0;        /* new parameter field encountered  */
            m = 0;
           data[++n1][m++] = string[n];
          }
       }
      else if (mode==2 && string[n]==']') mode = 1; /* end of a comment */
      else if (mode==3 && string[n]=='}') mode = 1; /* end of a comment */

    if (mode == 0)  data[n1][m] = 0;  /* place NULL chr at end of string */
     LOOP = FALSE;
```

```
    /*
      Exit the main loop only if the last character on the input line (other
      than a space or tab) is not &.  If it is, then read the next input line
      and repeat the loop, updating array DATA.
      Routine 'strrchr(st,ch)' returns the address of the last appearance of
      the character chr in string st (it returns 0 if ch does not appear in
       the string).
    */
    if ((loc=strrchr(&string[0],'&')) != 0) {
      for (++loc; *loc==' ' || *loc=='\t'; loc++) ; /* disregard spaces and
                                                    tabs after '&' */
      if (*loc == 0) {   /* last chr on line was '&'; read in new line */
         num = gets_1(s1);
         string = &s1[num];
         mode = 1;
         LOOP = TRUE;
       }
    }
  } while (LOOP == TRUE);

  /*
     Check the input to make sure it is in the correct form.  That is, the
     the cause must have either one or three fields, and there must be exactly
     six other fields present (three for effect, and one each for no., type,
     and precondition).  If the cause is a conjunction, then each '&' must be
     seperated by one or three fields.  If any of these restrictions does not
     hold, then set OK to FALSE.  Note that N1 indicates the total number of
     input fields, and the array DATA points to those fields.
  */

  OK = TRUE;
  num = 0;
  for (k1=1; (k1 < n1-4) && (OK == TRUE); k1++)
    if (data[k1][0] == '&')
       if (num != 1 && num != 3) OK = FALSE; /* must have 1 or 3 fields   */
       else num = 0;                         /* between each '&'          */
    else num = num + 1;
  if (num!=1 && num!=3) OK = FALSE;

  /*
    Generate output only if input was found to be in correct form.  First the
    appropriate cause output will be given, followed by the effect, and then
    by the prcondition (if not equal to "t").
  */

  if (OK == TRUE) {
   fprintf(fp2[i]," %s ",data[n1]); /* precondition field */
    if (k1 == 2) fprintf(fp2[i]," %s ",data[1]); /* 1 field, no '&' present */
    else if (k1==4 && data[2][0]!='&')      /* 3 fields, no '&' present */
      fprintf(fp2[i]," (%s %s %s) ",data[1],data[2],data[3]);
    else {  /* there is at least one '&' present  */
       k2 = 1;
       fprintf(fp2[i]," ; ("); /* print a semicolon to comment out the line */
```

```
      while (k2 < n1-4)
        if (data[k2+1][0] == '&') {  /* 1 field, followed by '&'  */
          fprintf(fp2[i],"%s & ",data[k2]);
           k2 = k2 + 2;
        }
       else if (data[k2+3][0] == '&') { /* 3 fields, followed by '&'  */
         fprintf(fp2[i],"(%s %s %s) & ",data[k2],data[k2+1],data[k2+2]);
           k2 = k2 + 4;
        }
       else if (k2+2 < n1-4) { /* last cause, 3 fields  */
         fprintf(fp2[i],"(%s %s %s) ",data[k2],data[k2+1],data[k2+2]);
           k2 = k2 + 3;
        }
       else {  /* last cause, 1 field  */
          fprintf(fp2[i],"%s ",data[k2]);
           k2 = k2 + 1;
        }
    }
    /* print rest of output line.  */
   fprintf(fp2[i],"(%s %s %s)\n",data[n1-4],data[n1-3],data[n1-2]);
  }
  return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                               *
 * GETS_1(STRING)                                                                *
 *                                                                               *
 * This routine reads a line of input and places it into the character array     *
 * STRING, replacing the newline charcater with a NULL character.  The           *
 * function returns the index of the first array element that is not a space     *
 * or a tab.  It returns '999' on end-of-file.                                   *
 *                                                                               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

gets_1(string)
char *string;
{
  int k;    /* number of spaces and tabs before first character */

  if (fgets(string,MAX_LENGTH,fp1) == NULL) return(999);
  for(k=0; string[k]==' ' || string[k]=='\t'; k++) ;
  *(&string[strcspn(string,"\n")]) = '\0';
  return(k);
}
```

Appendix H

AUTOMATIC CONVERSION PROGRAM FOR TEST DATABASE

```
/*****************************************************************
*                                                                *
* Program CON2                                                   *
*                                                                *
* This program converts FIS test data given in tabular form to data in a
* LISP format that can be used as input to the FIS package. The  *
* restrictions for the format of the input file are as follows:  *
*                                                                *
* 1. Nine columns of data are recognized. They are (in order): module name,
*    test point, parameter, units, possible qualitative values, maximum
*    value, minimum value, cost, and prequisites. Any column after these
*    nine is ignored.                                            *
* 2. Before the data are given, a column header must appear in order to
*    establish the column locations. This header should have the following
*    form:                                                       *
*                                    $3                          *
*            $1                      POSS                        *
*          TEST    $2                QUAL $4   $5   COST,        *
*    NAME  POINT  PARAMETER  UNITS  RES   MIN  MAX  SECONDS  PREREQUISITES
*                                                                *
*    Although preferable, the header does not have to look like this. CON2
*    simply looks at the lowest line in the header, and establishes column
*    locations by searching for groups of characters seperated by one or
*    more spaces. However, there must be exactly four lines in the header,
*    and the first line must contain only the characters "$3" (other than
*    spaces). If other columns are desired, there must be appropriate
*    column headers for them.                                    *
* 3. Comments may appear anywhere in the file. They are delimited on the
*    left by '[' and on the right by ']'. Anything appearing in the file
*    that is not part of a header is considered as data. Thus page numbers,
*    dates, page headers, etc. must be bracketed.                *
* 4. Data for a particular module do not have to appear on consecutive lines.
*    That is, comments and blank lines may break up a block of data. This is
*    useful, for example, if you would like to extend a prerequisite list
*    onto a next page. NOTE: A new column header may not break up a block
*    of data.                                                    *
* 5. The maximum number of characters allowed on an input line is
*    MAX_LENGTH  (specified in the C program).                   *
*                                                                *
* The output of the program is a series of lines in the foolowing form:
*                                                                *
*     (<name> <test_point> <parameter> <qual>                    *
*        p <units> (|cost| |prereq1| |prereq2| ... |prereq13|))  *
```

```
*                                                                              *
* <qual> may take several forms, depending on what is in the possible          *
* qualitative values column and what is in the min and max columns. This       *
* is summarized in the following chart:                                        *
*                                                                              *
*                  MIN and                                                     *
*   POSS QUAL   MAX given   <qual>                                             *
*   ---------------------------------------------------                        *
*   hi lo ok       YES      (ok (min max) hi (max inf) lo (-inf min))          *
*   hi lo ok       NO       (hi lo ok)                                         *
*   good lo ok     YES      (good (min max) hi (max inf) lo (-inf min))        *
*   good lo ok     NO       (good lo ok)                                       *
*   ok bad         YES      (ok (min max) bad (-inf min) (max inf))            *
*   ok bad         NO       (ok bad)                                           *
*   good bad       YES      (good (min max) bad (-inf min) (inf max))          *
*   good bad       NO       (good bad)                                         *
*   q1 q2 q3       YES      ERROR                                              *
*   q1 q2 q3       NO       (q1 q2 q3)                                         *
*   q1 q2          YES      ERROR                                              *
*   q1 q2          NO       (q1 q2)                                            *
*                                                                              *
* (q1, q2, and q3 indicate other qualitative descriptions)                     *
* For each block of test data there are thirteen prerequisites that must be    *
* specified (see routine 'print_pre' for a list of them). Each is assumed to   *
* have two parts: the prerequisite identifier and the variable descriptor      *
* (for example, 'active_mode on'). If the identifier is not found in the       *
* prerequisite list in the input, then that prerequisite will be given as |z|  *
* on the output.                                                               *
* When an error is encountered on the input, an error message will be          *
* printed at the line in which the error occured.                              *
*                                                                              *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

#include <stdio.h>
#define MAX_LENGTH 200     /* maximum length of input string  */
#define ER_MSG "***Error on this line*** "
#define ERROR ER1 = TRUE
#define ASSIGN(X) if (X[0] == '\0') strcpy(X,word); else ERROR; break
#define TRUE 1
#define FALSE 0

struct
{
 char terminal [30];
 char test [50][30];
}
 terminal_name[300];

int N, ER1;
FILE *fp1, *fp2,*fp3;

main()
```

```
{
  char string[MAX_LENGTH];  /* array into which input line is placed */
  char infile[30], outfile[30];  /* input, output files */
  int columns[MAX_LENGTH];  /* column identification array */
  char word[100], c[6];
  /* The following character arrays store the nine data fields for each block
     of test data. Note that pre1 is the is the first prerequisite field (the
     identifier) and pre2 the secound (the variable descriptor).
  */
  char name[30], test_point[30], parameter[30], min[10], max[10], cost[30],
      pre1[15][30], pre2[15][30], qual[3][10], units[30];
  int qual_num, pre_num;  /* array indexers for qual and pre1 and pre2 */
   int num, L, FIRST, i, k, a, j, error_line_count=0;


/* initalize the array */
i = 0;
j = 0;
for (i=0;i<300;++i)
 strcpy (terminal_name[i].terminal,"");
 for (j=0; j<=50; ++j)
  strcpy (terminal_name[i].test,"");

  ER1 = FALSE;  /* ER1=TRUE when an error on the input occurs */
  do {   /* Read in the input file name */
     printf("Enter the name of the input file... ");
     scanf("%s",infile);
     if ((fp1=fopen(infile,"r")) == NULL)
        printf("No such file exists.\n");
  } while (fp1 == NULL) ;


  printf("Do you wish the error file to be sent to the screen (s) or a file (f) ");
  scanf("%s",c);
  if (c[0]=='f' || c[0]=='F')
         fp3= fopen("ERROR_FILE","w");
  else
      fp3 = stdout;  /* stdout is the screen */
  printf("\n\n");


      /* Read in the output file name */
       printf("Enter the name of the output file... ");
       scanf("%s",outfile);

  printf("\n\n");

  FIRST = 1;

/*
  This is the main loop (highest level) loop in the program.  A line of input
  is read and stored in 'string'.  L indicates the length of the string and N
  is the current point in the string that is being examined.  Note that N is
  global and is updated in routine 'get_word', which returns the next word
```

```
  (group of characters) to 'word' and the column header number under which the
  word is located.
*/
  while (fgets(string,200,fp1) != NULL) {
    error_line_count++;
    string[strcspn(string,"\n")] = '\0';
    L = strlen(string);
    N = 0;
    while (N < L) {
     if (string[0]=='*')
     break; /* line begins with an * and should be ignored */
      num=get_word(word,string,columns);
      switch(num) {
        case 1: /* The current word is a module name. Print out data for
                   previous test data block, initialize pre_num, qual_num,
                   and FIRST, and set name equal to the current word.
                 */
                print_data(name,test_point,parameter,units,qual,min,max,
                        cost,pre1,pre2,pre_num,qual_num,FIRST,error_line_count);
                pre_num = -1;
                qual_num = -1;
                FIRST = 0;
                strcpy(name,word);
                break;
        case 2: ASSIGN(test_point); /* current word is test_point */
        case 3: ASSIGN(parameter); /* current word is parameter */
        case 4: ASSIGN(units);      /* current word is units     */
        case 5: /* Current word is one of the descriptors in the POSS QUAL
                   column. Increment the qaul_num index and store word
                   in the qual array.
                 */
                if (++qual_num > 2) ERROR;
                else strcpy(qual[qual_num],word);
                break;
        case 6: ASSIGN(min);        /* current word is the min value */
        case 7: ASSIGN(max);        /* current word is the max value */
        case 8: /* The current word is in the cost field. Since several
                   strings in a test data block may be assigned to this
                   field (because of constructs such as "10 + ping_cycle_
                   time") each word must be appended to the previous value
                   of cost with a space in between (unless, of course, the
                   current word ends with an underscore, in which case the
                   rest of that word is on the next line and the two must
                   be appended without the space).
                 */
                if (cost[0] == '\0') strcpy(cost,word);
                else {
                  k = strlen(cost);
                  if (cost[k-1] != '_') cost[k++] = ' ';
                  for (i=0; word[i]!='\0'; i++, k++)
                    cost[k] = word[i];
                  cost[k] = '\0';
                }
                break;
```

```
        case 9: /* The current word is the first part of a prerequisite
                   field. Increment the pre_num index and store it in the
                   pre1 array. Then get the next word. It should be the
                   second part of the prerequisite field; store it in pre2.
                */
                strcpy(pre1[++pre_num],word);
                if (get_word(word,string,columns) != 9) ERROR;
                else strcpy(pre2[pre_num],word);
                break;
        case 99: /* 99 indicates that the first row of a column header was
                    encountered. Read in the next three rows and call
                    routine 'find_col' to set up the column fields.
                 */
                for (k=1; k<=3; k++) fgets(string,MAX_LENGTH,fp1);
                find_col(string,columns);
        default: ;
      }
    }
  }
  /* The output for the last test data block has not been given yet. */
  print_data(name,test_point,parameter,units,qual,min,max,cost,pre1,pre2,
          pre_num,qual_num,FIRST,error_line_count);
  make_big_file(outfile); /* concatenates all the little files */
}

/*****************************************************************
 *                                                               *
 * GET_WORD(WORD,STRING,COLUMNS)                                 *
 *                                                               *
 * This routine receives an input line STRING, and it stores in WORD the *
 * first group of characters (the first word) at or after location N (N is the *
 * global variable indicating the position in the input line currently being *
 * examined). It also returns the number of the column in which the word *
 * appears. If the end of the string is reached then it returns -1, and if *
 * a column header is reached 99 is returned. COLUMNS is an array storing *
 * the locations of each of the columns.                         *
 *                                                               *
 *****************************************************************
*/

get_word(word,string,columns)
int columns[];
char *word, *string;
{
  char c;
  int k, k1, k2, m;
  int begin, end; /* begin is the location of the beginning of the word, and
                     end is the location of the end of the word. */

  /* Disregard all spaces and comments before the first character.
   */
```

```
  while (string[N]==' ' || string[N]=='[') {
    while (string[N] == ' ') N++;
    if (string[N] == '[') {
      while (string[N] != ']') N++;
       N++;
    }
  }

  if (string[N] == '\0') return(-1);  /* Return -1 if NUL character reached  */
  begin = N;
  k = 0;

  while (string[N]!=' ' && string[N]!=0)   /* Store group of characters */
     word[k++] = string[N++];              /* in word.                  */
  end = N - 1;
  word[k] = '\0';  /* Place NUL character at end of word.  */
  if (strcmp(word,"$3") == 0) {   /* Return 99 if column header found.  */
     for (k1=N; string[k1] == ' '; k1++) ;
     if (string[k1] == '\0') return(99);
  }

  /* The rest of the code in this routine determines in which column the word
     is located.  That column number is then returned.
  */

  if (columns[end] == columns[begin]) {
     if (columns[end] != 0) return(columns[end]);
      k = begin;
     while(columns[k]==columns[end] && k!=end) k++;
     if (k != end) return(columns[k]);
     for (k1=begin,k2=end; columns[k1]==columns[k2]; k1--,k2++) ;
     if (columns[k1] == columns[end]) return(columns[k2]);
     else return(columns[k1]);
  }
  for (k=begin; (columns[k]==columns[begin] || columns[k]==columns[end] ||
              columns[k]==0) && k!=end; k++) ;
  if (k != end) return(columns[k]);
  if (columns[begin] == 0) return(columns[end]);
  if (columns[end] == 0) return(columns[begin]);
  for (k1=begin,k2=end; columns[k1]==columns[begin] && columns[k2]==
                     columns[end]; k1++,k2--) ;
  if (columns[k2] == columns[end]) return(columns[end]);
  else return(columns[begin]);
}

/*****************************************************************************
 *                                                                           *
 * FIND_COL(STRING,COLUMNS)                                                  *
 *                                                                           *
 * This routine receives STRING, the lowest row in a column header, as input.*
 * It computes COLUMNS, an array in which each element contains a number     *
 * corresponding to the appropriate column that is loacted there. Columns are*
 * given increasing numbers starting at one, and those locations between     *
```

```
* columns are filled with 0's.  For example, the following column header                *
 * (STRING) generates COLUMNS as shown:                                                 *
 *                                                                                      *
 *  STRING = "   Header1         H2         Header3     Head4 "                         *
 * COLUMNS = "11111111110022000003333333004444440"                                      *
 *                                                                                      *
 * Note that leading spaces are filled with 1's.                                        *
 *                                                                                      *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

find_col(string,columns)
char *string;
int columns[];
{
  int k = 0, num = 0;

  while (string[k] == ' ') columns[k++] = 1;
  if (k == 0) columns[k++] = 1;
  while (string[k] != '\0') {
    if (string[k] == ' ') columns[k] = 0;
    else if (string[k-1] != ' ') columns[k] = num;
    else columns[k] = ++num;
    k++;
  }
  while (k < 200) columns[k++] = 0;
  return;
}
```

```
/*****************************************************************
*                                                                *
* IN(ST,QUAL,NUM)                                                *
*                                                                *
* This predicate function returns TRUE if the string ST is one of the strings *
* stored in the array of strings QUAL. There are NUM+1 strings in QUAL. *
*                                                                *
*****************************************************************
*/

in(st,qual,num)
char *st, qual[][10];
int num;
{
  int i;

  for (i=0; i<=num; i++)
   if (strcmp(qual[i],st) == 0) return(TRUE);
  return(FALSE);
}

/*****************************************************************
*                                                                *
* PRINT_DATA                                                     *
*                                                                *
* This routine prints the output for one block of test data and initializes *
* all test data variables. The format of the output is described in previous *
* documentation. If an error was found on the input (ER1=TRUE), then an *
* eror message is printed. On the first call to PRINT_DATA (FIRST=TRUE), *
* no data are printed.                                           *
*                                                                *
*****************************************************************
*/

print_data(name,test_point,parameter,units,qual,min,max,cost,pre1,pre2,
        pre_num,qual_num,FIRST,error_line_count)
char *name, *test_point, *parameter, qual[][10], *min, *max, pre1[][30],
   pre2[][30], *cost, *units;
int pre_num, qual_num, FIRST,error_line_count;
{
  int k,match=0,i=0, j, valid_test;
  char filename[30];
  if (!FIRST && !ER1)
  {
   while (strcmp(terminal_name[i].terminal,"") != 0)
      if (strcmp(test_point, terminal_name[i++].terminal)==0)
            {
            match =1;
            break;
            }
```

```
 if (strcmp(test_point,"")!=0)
      {
      strcpy(filename,test_point);
      fp2 = fopen (filename,"a+");
      }
 else
  return(-1);

  if (match == 0)
    {
    strcpy (terminal_name[i++].terminal,test_point);
     if (fp2 != NULL)
   fprintf(fp2," (%s\n",test_point);
      else
         printf ("FP2 IS NULL\n");
    }
 }

/* check to see if unique test */
valid_test = 1;
j=0;
while (strcmp(terminal_name[i-1].test[j],"") != 0)
 if (strcmp (terminal_name[i-1].test[j++],name)==0)
      {
       valid_test = 0;
      break;
      }


 if (valid_test)
 {
  strcpy(terminal_name[i-1].test[j],name);
  if (!FIRST && !ER1 ) {
    fprintf(fp2,"   (%s %s\n",name,parameter);
    fprintf(fp2,"          S1\n"); /*just a guess */
    if (qual_num==1 && min[0]=='\0')
      fprintf(fp2,"          ((%s) (%s))\n",qual[1],qual[0]); /*get into format (ok bad) */
    else if (qual_num==2 && min[0]=='\0')
      fprintf(fp2,"          ((%s)(%s)(%s))\n",qual[2],qual[1],qual[0]); /*ok lo hi */
    else if (qual_num==1)
      if (in("bad",qual,1) && in("ok",qual,1))
        fprintf(fp2,"          ((OK (%s %s)) (BAD (-INF %s) (%s INF))) \n", min,max,min,max);
      else if (in("bad",qual,1) && in("good",qual,1))
        fprintf(fp2,"          ((GOOD (%s %s)) (BAD (-INF %s) (%s INF)))\n",
                min,max,min, max);
       else ERROR;
    else if (qual_num==2)
      if (in("hi",qual,2) && in("lo",qual,2) && in("ok",qual,2))
 {
       fprintf(fp2,"          ((OK ((%s %s))) (LO ((-inf %s)))\n",
              min,max,min);
        fprintf(fp2,"                          (HI ((%s inf))))\n",
              max);
        }
```

```
      else if (in("hi",qual,2) && in("lo",qual,2) && in("good",qual,2))
        {
        fprintf(fp2,"          ((GOOD (%s %s))) (LO ((-INF %s)))\n",
            min,max,min);
        fprintf(fp2,"                              (HI ((%s INF))))\n",
            max);
        }
      else ERROR;
    else ERROR;
    fprintf(fp2,"          %s\n",units);
    fprintf(fp2,"          D\n");
    fprintf(fp2,"          10\n");
    fprintf(fp2,"          NIL)\n");
  }
}

  /* Print error message if ER1=TRUE.  */
  if (ER1) {
    fprintf(fp3,ER_MSG);
   fprintf(fp3,"#%d\n",error_line_count);
    ER1 = FALSE;
  }

  /* Initialize test data variables  */

  test_point[0] = '\0';
  parameter[0] = '\0';
  units[0] = '\0';
  min[0] = '\0';
  max[0] = '\0';
  cost[0] = '\0';
  for (k=0; k<=2; k++) qual[k][0] = '\0';
  for (k=0; k<=14; k++) {
    pre1[k][0] = '\0';
    pre2[k][0] = '\0';
  }
  fclose (fp2);
  return;
}

/*****************************************************************************
 *                                                                           *
 * PRINT_PRE                                                                 *
 *                                                                           *
 * This routine prints the preconditions and cost field (format described in *
 * previous documentation). The thirteen preconditions are defined in array  *
 * strings (note the zeroth element is a dummy string)                       *
 *                                                                           *
 *****************************************************************************
*/

print_pre(pre1,pre2,pre_num,cost)
char pre1[][30], pre2[][30], *cost;
int pre_num;
```

```
{
  int n, n1, LOOP;
  static char strings[14][15] = {
    "zzz", "active_mode", "unit-26_door", "26A1_drawer", "unit-34", "scope",
    "unit-34_brg", "probe_on", "ac_diff_mtr", "counter", "diff_scope",
    "diff_counter", "PMFL_f", "PMFL_s"
  };

  /* Print the cost field   */

  if (cost[0] != '\0')
    fprintf(fp3,"(!%s!",cost);
  else
    fprintf(fp3,"(!z!");

  /* Print each of the thirteen preconditions   */

  for (n=1; n<=13; n++) {
    LOOP = TRUE;
    /* Find a precondition that matches strings[n]   */
    for (n1=0; n1<=pre_num && LOOP==TRUE; n1++)
      if (strcmp(pre1[n1],strings[n]) == 0) LOOP = FALSE;
    n1--;
   if (n==2 || n==6 || n==10)
      fprintf(fp3,"\n    ");
    if (LOOP == FALSE)  /* A precondition was found that matches string[n] */
      fprintf(fp3," !%s %s!",pre1[n1],pre2[n1]);
    else    /* LOOP = TRUE; a precondition was not found     */
      fprintf(fp3," !z!");
  }
}



make_big_file(outfile)
char *outfile;

{
char names2[50],string[1000],old_file[50];
int i=0,one=0;

while(strcmp(terminal_name[i].terminal,"")!=NULL)
  {
  if (one)
   strcpy(names2,"HUMPTY");
  else
   strcpy(names2,"dumpty");
  strcpy(string,"");
  strcpy(string,"cat ");
  strcat(string,old_file);
  strcat(string," ");
  strcat(string,terminal_name[i].terminal);
  strcat(string," > ");
  strcat(string, names2);
```

```
 strcpy(old_file,names2);
 system(string);
 if (one ==0)
    one =1;
 else
    one = 0;
 ++i;
 }
/* copy to the user defined outfile name */
strcpy(string,"");
strcpy(string,"cp ");
if (!one)
  strcat(string," HUMPTY ");
 else
  strcat(string," dumpty ");

 strcat(string,outfile);
 system(string);
}
```

Appendix I

SEMIAUTOMATIC CONVERSION PROGRAM FOR TEST DATABASE

```
#include <stdio.h>
#define MAX_LENGTH 200     /* maximum length of input string */
#define ER_MSG "***Error on this line***"
#define ERROR ER1 = TRUE
#define ASSIGN(X) if (X[0] == '\0') strcpy(X,word); else ERROR; break
#define TRUE 1
#define FALSE 0

struct
{
 char terminal [30];
 char test [50][30];
}
 terminal_name[300];

int N, ER1;
FILE *fp1, *fp2,*fp3;

main()
{
  char string[MAX_LENGTH];  /* array into which input line is placed */
  char infile[30], outfile[30];  /* input, output files */
  int columns[MAX_LENGTH];  /* column identification array */
  char word[100], c[6];
  /* The following character arrays store the nine data fields for each block
     of test data. Note that pre1 is the first prerequisite field (the
     identifier) and pre2 the second (the variable descriptor).
  */
  char name[30], test_point[30], parameter[30], min[10], max[10], cost[30],
      pre1[15][30], pre2[15][30], qual[3][10], units[30];
  int qual_num, pre_num;  /* array indexers for qual and pre1 and pre2 */
   int num, L, FIRST, i, k, a, j, error_line_count=0;


/* initalize the array */
i = 0;
j = 0;
for (i=0;i<300;++i)
 strcpy (terminal_name[i].terminal,"");
 for (j=0; j<=50; ++j)
  strcpy (terminal_name[i].test,"");
  ER1 = FALSE;  /* ER1=TRUE when an error on the input occurs */
```

```
  do {  /* Read in the input file name */
    printf("Enter the name of the input file... ");
    scanf("%s",infile);
   if ((fp1=fopen(infile,"r")) == NULL)
      printf("No such file exists.\n");
  } while (fp1 == NULL) ;
 printf("Do you wish the output to be sent to the screen (s) or a file (f) ");
 scanf("%s",c);
 if (c[0] == 'f')
    do {  /* Read in the output file name */
      printf("Enter the name of the output file... ");
      scanf("%s",outfile);
      if ((a=access(outfile,0)) == 0) {
         printf("The file already exists.\n");
         printf("Do you wish to write over it (y or n)? ");
         scanf("%s",c);
       }
      if ((a==0 && (c[0]=='y' || c[0]=='Y')) || a!=0)
         fp3= fopen(outfile,"w");
    } while (a==0 && (c[0]=='n' || c[0]=='N'));
 else fp3 = stdout;  /* stdout is the screen */
 printf("\n\n");

 FIRST = 1;

/*
  This is the main loop (highest level) loop in the program.  A line of input
  is read and stored in 'string'.  L indicates the length of the string and N
  is the current point in the string that is being examined.  Note that N is
  global and is updated in routine 'get_word', which returns the next word
  (group of characters) to 'word' and the column header number under which the
  word is located.
*/
  while (fgets(string,200,fp1) != NULL) {
    error_line_count++;
    string[strcspn(string,"\n")] = '\0';
    L = strlen(string);
    N = 0;
    while (N < L) {
      if (string[0] == '*')
        break; /* erroneous line */
      num=get_word(word,string,columns);
      switch(num) {
        case 1:  /* The current word is a module name.  Print out data for
                    previous test data block, initialize pre_num, qual_num,
                    and FIRST, and set name equal to the current word.
                  */
                 print_data(name,test_point,parameter,units,qual,min,max,
                         cost,pre1,pre2,pre_num,qual_num,FIRST,error_line_count);
                 pre_num = -1;
                 qual_num = -1;
                 FIRST = 0;
                 strcpy(name,word);
                 break;
```

```
            case 2:  ASSIGN(test_point);  /* current word is test_point */
            case 3:  ASSIGN(parameter);   /* current word is parameter  */
            case 4:  ASSIGN(units);       /* current word is units      */
            case 5:  /* Current word is one of the descriptors in the POSS QUAL
                        column.  Increment the qaul_num index and store word
                        in the qual array.
                     */                  if (++qual_num > 2) ERROR;
                     else strcpy(qual[qual_num],word);
                     break;
            case 6:  ASSIGN(min);         /* current word is the min value */
            case 7:  ASSIGN(max);         /* current word is the max value */
            case 8:  /* The current word is in the cost field.  Since several
                        strings in a test data block may be assigned to this
                        field (because of constructs such as "10 + ping_cycle_
                        time") each word must be appended to the previous value
                        of cost with a space in between (unless, of course, the
                        current word ends with an underscore, in which case the
                        rest of that word is on the next line and the two must
                        be appended without the space).
                     */
                     if (cost[0] == '\0') strcpy(cost,word);
                     else {
                       k = strlen(cost);
                       if (cost[k-1] != '_') cost[k++] = ' ';
                       for (i=0; word[i]!='\0'; i++, k++)
                         cost[k] = word[i];
                       cost[k] = '\0';
                     }
                     break;
            case 9:  /* The current word is the first part of a prerequisite
                        field.  Increment the pre_num index and store it in the
                        pre1 array.  Then get the next word.  It should be the
                        second part of the prerequisite field; store it in pre2.
                     */
                     strcpy(pre1[++pre_num],word);
                     if (get_word(word,string,columns) != 9) ERROR;
                     else strcpy(pre2[pre_num],word);
                     break;
            case 99: /* 99 indicates that the first row of a column header was
                        encountered.  Read in the next three rows and call
                        routine 'find_col' to set up the column fields.
                     */
                     for (k=1; k<=3; k++) fgets(string,MAX_LENGTH,fp1);
                     find_col(string,columns);
            default: ;
          }
        }
      }
      /* The output for the last test data block has not been given yet.  */
      print_data(name,test_point,parameter,units,qual,min,max,cost,pre1,pre2,
              pre_num,qual_num,FIRST,error_line_count);
    }
```

```
/* ********************************************************************** *
 *                                                                        *
 * GET_WORD(WORD,STRING,COLUMNS)                                          *
 *                                                                        *
 * This routine receives an input line STRING, and it stores in WORD the first *
 * group of characters (the first word) at or after location N (N is the *
 * global variable indicating the position in the input line currently being *
 * examined). It also returns the number of the column in which the word  *
 * appears. If the end of the string is reached then it returns -1, and if *
 * a column header is reached 99 is returned. COLUMNS is an array storing the *
 * locations of each of the columns.                                      *
 *                                                                        *
 * ********************************************************************** *
*/

get_word(word,string,columns)
int columns[];
char *word, *string;
{
  char c;
   int k, k1, k2, m;
  int begin, end;  /* begin is the location of the beginning of the word, and
                      end is the location of the end of the word.  */

  /* Disregard all spaces and comments before the first character.
   */
  while (string[N]==' ' || string[N]=='[') {
     while (string[N] == ' ') N++;
     if (string[N] == '[') {
        while (string[N] != ']') N++;
        N++;
     }
  }

  if (string[N] == '\0') return(-1);  /* Return -1 if NUL character reached  */
  begin = N;
  k = 0;

  while (string[N]!=' ' && string[N]!=0)   /* Store group of characters */
     word[k++] = string[N++];              /* in word.                  */

  end = N - 1;
  word[k] = '\0';  /* Place NUL character at end of word.  */
  if (strcmp(word,"$3") == 0) {   /* Return 99 if column header found.  */
     for (k1=N; string[k1] == ' '; k1++) ;
     if (string[k1] == '\0') return(99);
  }

  /* The rest of the code in this routine determines in which column the word
     is located.  That column number is then returned
   */

  if (columns[end] == columns[begin]) {
     if (columns[end] != 0) return(columns[end]);
```

```
    k = begin;
    while(columns[k]==columns[end] && k!=end) k++;
    if (k != end) return(columns[k]);
    for (k1=begin,k2=end; columns[k1]==columns[k2]; k1--,k2++) ;
    if (columns[k1] == columns[end]) return(columns[k2]);
    else return(columns[k1]);
  }
  for (k=begin; (columns[k]==columns[begin] || columns[k]==columns[end] ||
            columns[k]==0) && k!=end; k++) ;
  if (k != end) return(columns[k]);
  if (columns[begin] == 0) return(columns[end]);
  if (columns[end] == 0) return(columns[begin]);
  for (k1=begin,k2=end; columns[k1]==columns[begin] && columns[k2]==
                columns[end]; k1++,k2--) ;
  if (columns[k2] == columns[end]) return(columns[end]);
  else return(columns[begin]);
}

/* ********************************************************** **
 *                                                            *
 * FIND_COL(STRING,COLUMNS)                                   *
 *                                                            *
 * This routine receives STRING, the lowest row in a column header, as input.  *
 * It computes COLUMNS, an array in which each element contains a number       *
 * corresponding to the appropriate column that is located there. Columns are  *
 * given increasing numbers starting at one, and those locations between       *
 * columns are filled with 0's. For example, the following column header       *
 * (STRING) generates COLUMNS as shown:                       *
 *                                                            *
 *  STRING = "   Header1  H2     Header3  Head4 "             *
 * COLUMNS = "11111111110022000003333333004444440"            *
 *                                                            *
 * Note that leading spaces are filled with 1's.              *
 *                                                            *
 * ********************************************************** **
*/

find_col(string,columns)
char *string;
int columns[];
{
  int k = 0, num = 0;

  while (string[k] == ' ') columns[k++] = 1;
  if (k == 0) columns[k++] = 1;
  while (string[k] != '\0') {
    if (string[k] == ' ') columns[k] = 0;
    else if (string[k-1] != ' ') columns[k] = num;
    else columns[k] = ++num;
    k++;
  }
  while (k < 200) columns[k++] = 0;
  return;
}
```

```
/* - ***********************************************************
 *                                                            *
 * IN(ST,QUAL,NUM)                                            *
 *                                                            *
 * This predicate function returns TRUE if the string ST is one of the strings   *
 * stored in the array of strings QUAL.  There are NUM+1 strings in QUAL.         *
 *                                                            *
 ***************************************************************
*/

in(st,qual,num)
char *st, qual[][10];
int num;
{
  int i;

  for (i=0; i<=num; i++)
   if (strcmp(qual[i],st) == 0) return(TRUE);
  return(FALSE);
}

/**************************************************************
 *                                                            *
 * PRINT_DATA                                                 *
 *                                                            *
 * This routine prints the output for one block of test data and initializes   *
 * all test data variables.  The format of the output is described in previous  *
 * documentation.  If an error was found on the input (ER1=TRUE), then an       *
 * error message is printed.  On the first call to PRINT_DATA (FIRST=TRUE), no  *
 * data are printed.                                          *
 *                                                            *
 ***************************************************************
*/

print_data(name,test_point,parameter,units,qual,min,max,cost,pre1,pre2,
        pre_num,qual_num,FIRST,error_line_count)
char *name, *test_point, *parameter, qual[][10], *min, *max, pre1[][30],
    pre2[][30], *cost, *units;
int pre_num, qual_num, FIRST,error_line_count;
{
  int k,match=0,i=0, j, valid_test;
  char filename[30];
  if (!FIRST && !ER1)
  {
   while (strcmp(terminal_name[i].terminal,"") != 0)
      if (strcmp(test_point, terminal_name[i++].terminal)==0)
             {
             match =1;
             break;
             }
   if (strcmp(test_point,"")!=0)
       {
        strcpy(filename,test_point);
```

```
        fp2 = fopen (filename,"a+");
        }
    else
     return(-1);    if (match == 0)
       {
       strcpy (terminal_name[i++].terminal,test_point);
        if (fp2 == NULL)
           printf ("FP2 IS NULL\n");
       }
    }

/* check to see if unique test */
valid_test = 1;
j=0;
while (strcmp(terminal_name[i-1].test[j],"") != 0)
 if (strcmp (terminal_name[i-1].test[j++],name)==0)
       {
        valid_test = 0;
       break;
       }


if (valid_test)
{
 strcpy(terminal_name[i-1].test[j],name);
 if (!FIRST && !ER1 ) {
   fprintf(fp2,"%s %s s1 ",name,parameter);
   if (qual_num==1 && min[0]=='\0')
     fprintf(fp2,"((%s) (%s)) "  ,qual[1],qual[0]);
   else if (qual_num==2 && min[0]=='\0')
    fprintf(fp2,"((%s)(%s)(%s)) ",qual[2],qual[1],qual[0]);
   else if (qual_num==1)
     if (in("bad",qual,1) && in("ok",qual,1))
       fprintf(fp2,"((OK (%s %s)) (BAD (-INF %s) (%s INF))) ", min,max,min,max);
     else if (in("bad",qual,1) && in("good",qual,1))
       fprintf(fp2,"((GOOD (%s %s)) (BAD (-INF %s) (%s INF))) ",
                  min,max,min, max);
      else ERROR;
   else if (qual_num==2)
     if (in("hi",qual,2) && in("lo",qual,2) && in("ok",qual,2))
       {
      fprintf(fp2,"((OK ((%s %s))) (LO ((-inf %s))) ",
              min,max,min);
       fprintf(fp2,"(HI ((%s inf)))) ",
              max);
       }
     else if (in("hi",qual,2) && in("lo",qual,2) && in("good",qual,2))
       {
      fprintf(fp2,"((GOOD (%s %s))) (LO ((-INF %s)))",
           min,max,min);
       fprintf(fp2,"(HI ((%s INF)))) ",
              max);
       }
      else ERROR;
```

```
      else ERROR;
      fprintf(fp2,"%s ",units);
      fprintf(fp2,"diag 10 nil\n");
    }
  }

  /* Print error message if ER1=TRUE.  */
  if (ER1) {
     fprintf(fp3,ER_MSG);
    fprintf(fp3," #%d\n",error_line_count);
     ER1 = FALSE;
  }

  /* Initialize test data variables  */

  test_point[0] = '\0';
  parameter[0] = '\0';
  units[0] = '\0';
  min[0] = '\0';
  max[0] = '\0';
  cost[0] = '\0';
  for (k=0; k<=2; k++) qual[k][0] = '\0';
  for (k=0; k<=14; k++) {
     pre1[k][0] = '\0';
     pre2[k][0] = '\0';
  }
  fclose (fp2);
  return;
}

/* ************************************************************** *
 *                                                                *
 * PRINT_PRE                                                      *
 *                                                                *
 * This routine prints the preconditions and cost field (format described in  *
 * previous documentation). The thirteen preconditions are defined in array   *
 * strings (note the zeroth element is a dummy string)            *
 *                                                                *
 * ************************************************************** *
*/

print_pre(pre1,pre2,pre_num,cost)
char pre1[][30], pre2[][30], *cost;
int pre_num;
{
  int n, n1, LOOP;
  static char strings[14][15] = {
    "zzz", "active_mode", "unit-26_door", "26A1_drawer", "unit-34", "scope",
    "unit-34_brg", "probe_on", "ac_diff_mtr", "counter", "diff_scope",
    "diff_counter", "PMFL_f", "PMFL_s"
  };

  /* Print the cost field   */
```

```
  if (cost[0] != '\0')
    fprintf(fp3,"(|%s|",cost);
  else
    fprintf(fp3,"(|z|");

  /* Print each of the thirteen preconditions  */

  for (n=1; n<=13; n++) {
    LOOP = TRUE;
    /* Find a precondition that matches strings[n]  */
    for (n1=0; n1<=pre_num && LOOP==TRUE; n1++)
      if (strcmp(pre1[n1],strings[n]) == 0) LOOP = FALSE;
    n1--;
    if (n==2 || n==6 || n==10)
      fprintf(fp3,"\n    ");
    if (LOOP == FALSE)  /* A precondition was found that matches string[n] */
      fprintf(fp3," |%s %s|",pre1[n1],pre2[n1]);
    else    /* LOOP = TRUE; a precondition was not found    */
      fprintf(fp3," |z|");
  }
}
```

Appendix J

TEST DATABASE INSTRUCTION INDEX PROGRAM

```
/*

  CON3

  This program converts test data given in tabular form to a LISP list defined
  by the LISP function 'instr-list'. The format of the input file is the same
  as that described in CON2, except that two more columns are recognized.
  Column ten is the instruction name and column 11 the text parameters. The
  headers should look as follows:

                  . . .                 INSTR      TEXT
  NAME                                  NAME     PARAMETERS

  The LISP function generated has the following format:

  (defun instr-list ()
      '(
        (test_point1
        (parameter instr_name (min max) record [text_par1] [text_par2])
        (parameter instr_name (min max) record [text_par1] [text_par2])
                          :
        (test_point2
                          :
      ) )

  Note that the text parameters are optional and appear only if given on the
  input.  If min and max are not given on the input, then '(nil nil)' will
  substitute min and max.
  The list is arranged such that all test data with the same test point are
  grouped as sublists under that test point.  This allows for a quicker lookup
  in the LISP searching routines.
  Program flow and execution are very similar to CON2, except that after each
  data block has been read in, it must be stored instead of immediately being
  printed out because of the ordering as described.  Because of the possible
  size of the input file, a linked list structure is used, thereby minimizing
  necessary memory.


  NOTE: this program needs two input files.  One which he describes and
        another which lists all of the testnames (testlist.all).
*/
```

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
#define MAX_LENGTH 200     /* maximum length of input string */
#define ER_MSG "***Error on this line***\n"
#define ERROR ER1 = TRUE
#define ASSIGN(X) if (X[0] == '\0') strcpy(X,word); else ERROR; break
#define STR_CPY(S) strcpy((S)->parameter,parameter); strcpy((S)->min,min); \
      strcpy((S)->max,max); strcpy((S)->text1,&text1[3]); \
      strcpy((S)->text2,&text2[3]),strcpy((S)->instr,instr); (S)->p2 = NULL
#define RECSIZE 512
/* the following 3 constants indicate the maximum size of a parameter to a
   test instruction. */
#define INSTNAMESIZE 30
#define MINMAXSIZE 10
#define TEXTSIZE 30
struct s1 {
  char test_point[INSTNAMESIZE];
  struct s1 *p1;
  struct s2 *p2;
};
struct s2 {
  char parameter[INSTNAMESIZE];
  char min[MINMAXSIZE];
  char max[MINMAXSIZE];
  char text1[TEXTSIZE];
  char text2[TEXTSIZE];
  char instr[INSTNAMESIZE];
  struct s2 *p2;
};
/*  This structure keeps track of all the instruction names */
struct inst_name_struct {
 char instr[INSTNAMESIZE];
  int position;
 struct inst_name_struct *next;
} *test_name_head; /* the head of the instruction list. Used by get_record */
int N, ER1;
FILE *fp1, *fp2, *fp3;
char infile[30], outfile[30], instruction[30];   /* input, output,   instruction files */

main()
{
  char string[MAX_LENGTH];   /* array into which input line is placed */
  int columns[MAX_LENGTH];   /* column identification array */
  char word[133], c[MINMAXSIZE];
  char test_point[INSTNAMESIZE], parameter[INSTNAMESIZE], min[MINMAXSIZE],
   max[MINMAXSIZE], instr[INSTNAMESIZE], text1[TEXTSIZE], text2[TEXTSIZE];
   int num, L, FIRST, LOOP, i, k, a, CONT;
   struct s1 *head, *tp, *tp1, *tp2;
   struct s2 *par, *par1;

  test_name_head=NULL;

  ER1 = FALSE;  /* ER1=TRUE when an error on the input occurs */
```

```
do {  /* Read in the input file name */
  printf("Enter the name of the input file... ");
  scanf("%s",infile);
  if ((fp1=fopen(infile,"r")) == NULL)
     printf("No such file exists.\n");
} while (fp1 == NULL) ;

do {  /* Read in the instruction file name */
  printf("Enter the name of the instruction file... ");
  scanf("%s",instruction);
  if ((fp3=fopen(instruction,"r")) == NULL)
     printf("No such file exists.\n");
} while (fp3 == NULL) ;

printf("Do you wish the output to be sent to the screen (s) or a file (f) ");
scanf("%s",c);
if (c[0] == 'f')
  do {  /* Read in the output file name */
     printf("Enter the name of the output file... ");
     scanf("%s",outfile);
    if ((a=access(outfile,0)) == 0) {
       printf("The file already exists.\n");
       printf("Do you wish to write over it (y or n)? ");
       scanf("%s",c);
     }
    if ((a==0 && (c[0]=='y' || c[0]=='Y')) || a!=0)
       fp2 = fopen(outfile,"w");
  } while (a==0 && (c[0]=='n' || c[0]=='N'));
else fp2 = stdout;  /* stdout is the screen */
printf("\n\n");

FIRST = TRUE;
head = NULL;
CONT = TRUE;
while (CONT == TRUE) {
  if (fgets(string,MAX_LENGTH,fp1) == NULL) {
     strcpy(string,"***LAST***");
     CONT = FALSE;
   }
  string[strcspn(string,"\n")] = '\0';
  L = strlen(string);
  N = 0;
  while (N < L) {
    num=get_word(word,string,columns);
    switch(num) {
       case 1:  if (!FIRST) {
                  tp1 = head;
                  LOOP = TRUE;
                for (tp=head; tp!=NULL && LOOP==TRUE; tp=tp->p1)
                   if (strcmp(tp->test_point,test_point) == 0) {
                    for (par=tp->p2; par->p2 != NULL; par=par->p2);
                     par1 =(struct s2 *) malloc (sizeof (struct s2));
                      par->p2 = par1;
                      STR_CPY(par1);
```

```
                    LOOP = FALSE;
                  }
                  else tp1 = tp;
                if (LOOP) {
                  tp2 = (struct s1 *)malloc (sizeof (struct s1));
                  if (tp1 == NULL) head = tp2;
                   else tp1->p1 = tp2;
                   strcpy(tp2->test_point,test_point);
                   tp2->p1 = NULL;
                  par = (struct s2 *)malloc (sizeof (struct s2));
                   tp2->p2 = par;
                   STR_CPY(par);
                 }
               }
              parameter[0] = '\0';
               min[0] = '\0';
              max[0] = '\0';
              text1[3] = '\0';
              text2[3] = '\0';
              instr[0] = '\0';
              test_point[0] = '\0';
               FIRST = FALSE;
              break;
      case 2:  ASSIGN(test_point);  /* current word is test_point */
      case 3:  ASSIGN(parameter);   /* current word is parameter  */
      case 6:  ASSIGN(min);         /* current word is the min value */
      case 7:  ASSIGN(max);         /* current word is the max value */
      case 10: ASSIGN(instr);
      case 11: if (text1[3] == '\0') strcpy(text1,word);
               else strcpy(text2,word);
               break;
      case 99: /* 99 indicates that the first row of a column header was
                  encountered.  Read in the next three rows and call
                  routine 'find_col' to set up the column fields.
                */
               for (k=1; k<=3; k++) fgets(string,MAX_LENGTH,fp1);
               find_col(string,columns);
      default: ;
    }
  }
}

fprintf(fp2,"(defun instr-list ()\n     '(\n");
for (tp=head; tp != NULL; tp=tp->p1) {
 fprintf(fp2,"        (%s",tp->test_point);
 for (par=tp->p2; par != NULL; par=par->p2) {
   fprintf(fp2,"\n          (%s %s ",par->parameter,      par->instr);
   if (par->min[0]=='\0')  fprintf(fp2,"(nil nil)");
   else fprintf(fp2,"(%s %s)",par->min,par->max);
   fprintf(fp2," %d",get_record(par->instr));    /* indicates where the */
                                                 /* instruction text is */
   if (par->text1[0] != '\0') fprintf(fp2," %s",par->text1);
   if (par->text2[0] != '\0') fprintf(fp2," %s",par->text2);
   fputc(')',fp2);
```

```
    }
    fprintf(fp2,")\n");
     fprintf(fp2,")      )");

}

/* ********************************************************************** *
 *                                                                        *
 * GET_WORD(WORD,STRING,COLUMNS)                                          *
 *                                                                        *
 * This routine receives an input line STRING, and it stores in WORD the first *
 * group of characters (the first word) at or after location N (N is the  *
 * global variable indicating the position in the input line currently being *
 * examined). It also returns the number of the column in which the word  *
 * appears. If the end of the string is reached then it returns -1, and if *
 * a column header is reached 99 is returned. COLUMNS is an array storing the *
 * locations of each of the columns.                                      *
 *                                                                        *
 * ********************************************************************** *
*/

get_word(word,string,columns)
int columns[];
char *word, *string;
{
  char c;
   int k, k1, k2, m;
  int begin, end;  /* begin is the location of the beginning of the word, and
                      end is the location of the end of the word.  */

  /* Disregard all spaces and comments before the first character.
   */
  while (string[N]==' ' || string[N]=='[') {
    while (string[N] == ' ') N++;
    if (string[N] == '[') {
      while (string[N] != ']') N++;
      N++;
    }
  }

  if (string[N] == '\0') return(-1);  /* Return -1 if NUL character reached  */
  begin = N;
  k = 0;

  while (string[N]!=' ' && string[N]!=0)   /* Store group of characters */
    word[k++] = string[N++];               /* in word.                  */

  end = N - 1;
  word[k] = '\0';  /* Place NUL character at end of word.  */
  if (strcmp(word,"$3") == 0) {   /* Return 99 if column header found.  */
    for (k1=N; string[k1] == ' '; k1++) ;
    if (string[k1] == '\0') return(99);
  }
  if (strcmp(word,"***LAST***") == 0) return(1);
```

```
  /* The rest of the code in this routine determines in which column the word
    is located. That column number is then returned.
  */

  if (columns[end] == columns[begin]) {
    if (columns[end] != 0) return(columns[end]);
     k = begin;
    while(columns[k]==columns[end] && k!=end) k++;
    if (k != end) return(columns[k]);
    for (k1=begin,k2=end; columns[k1]==columns[k2]; k1--,k2++) ;
    if (columns[k1] == columns[end]) return(columns[k2]);
    else return(columns[k1]);
  }
  for (k=begin; (columns[k]==columns[begin] || columns[k]==columns[end] ||
            columns[k]==0) && k!=end; k++) ;
  if (k != end) return(columns[k]);
  if (columns[begin] == 0) return(columns[end]);
  if (columns[end] == 0) return(columns[begin]);
  for (k1=begin,k2=end; columns[k1]==columns[begin] && columns[k2]==
                  columns[end]; k1++,k2--) ;
  if (columns[k2] == columns[end]) return(columns[end]);
  else return(columns[begin]);
}

/*****************************************************************
 *                                                               *
 * FIND_COL(STRING,COLUMNS)                                      *
 *                                                               *
 * This routine receives STRING, the lowest row in a column header, as input *
 * It computes COLUMNS, an array in which each element contains a number *
 * corresponding to the appropriate column that is located there. Columns are *
 * given increasing numbers starting at one, and those locations between *
 * columns are filled with 0's. For example, the following column header *
 * (STRING) generates COLUMNS as shown:                          *
 *                                                               *
 *  STRING = "   Header1  H2     Header3  Head4 "                *
 * COLUMNS = "1111111111002200000333333300444440"                *
 *                                                               *
 * Note that leading spaces are filled with 1's.                 *
 *                                                               *
 *****************************************************************
*/

find_col(string,columns)
char *string;
int columns[];
{
  int k = 0, num = 0;
  while (string[k] == ' ') columns[k++] = 1;
  if (k == 0) columns[k++] = 1;
  while (string[k] != '\0') {
    if (string[k] == ' ') columns[k] = 0;
    else if (string[k-1] != ' ') columns[k] = num;
```

```
    else columns[k] = ++num;
     k++;
  }
  while (k < 200) columns[k++] = 0;
  return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *                                                                                              *
 * IN(ST,QUAL,NUM)                                                                              *
 *                                                                                              *
 * This predicate function returns TRUE if the string ST is one of the strings                  *
 * stored in the array of strings QUAL. There are NUM+1 strings in QUAL.                        *
 *                                                                                              *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *8
*/

in(st,qual,num)
char *st, qual[][10];
int num;
{
  int i;

  for (i=0; i<=num; i++)
   if (strcmp(qual[i],st) == 0) return(TRUE);
  return(FALSE);
};

/*****************************************************************
*
* get_inst_location does a search for a given instruction number, inst_name,
*      and returns it's record number in the instructions file. If it
*      does not exist, a message is printed indicating so. Record numbers
*      start at zero.
*
*****************************************************************/

int get_inst_location(inst_name)
char *inst_name;
{
 int inst_rec;
 char read_inst_name[INSTNAMESIZE];
 inst_rec=0;
 do {
  fseek(fp3,inst_rec*RECSIZE,0);
   inst_rec++;
 } while ((fscanf(fp3,"%s",read_inst_name)!=EOF) && (strcmp(inst_name,read_inst_name)!=0));
  if (strcmp(read_inst_name,inst_name)!=0) {
    printf("The instruction to %s, a test in %s,\n",inst_name,infile);
    printf("has no corresponding instruction in %s\n",instruction);
   }
 --inst_rec;
 return(inst_rec);
}
```

```
/*********************************************************************
*
* get_record returns the record number of a given instruction name inst_name.
*       It builds a list of instructions that it has found, so that it doesn't
*       have to search the disk twice for the same instruction number.
*       The list is built as each new instruction comes along.
*********************************************************************/

int get_record(inst_name)
char *inst_name;
{
 struct inst_name_struct *list; /* current instruction pointer          */
 int rec_num;                   /* returned value from get_inst_location */
 /* this test is true if this is the first time this routine is called  */
 /* if it is the first time, search the disk for the instruction num.   */
 if (test_name_head == NULL) {
  test_name_head=(struct inst_name_struct *)
    malloc (sizeof (struct inst_name_struct));
  strcpy(test_name_head->instr,inst_name);
  test_name_head->next=NULL;
  test_name_head->position=get_inst_location(inst_name);
  return(test_name_head->position);
 };

 /* look for the instruction in the list */
 list=test_name_head;
 while ((strcmp(inst_name,list->instr)!=0) && (list->next != NULL))
   list=list->next;
 if (strcmp(inst_name,list->instr)==0) return (list->position);

 /* not in list, add it to the end */
 list->next=(struct inst_name_struct *)
  malloc (sizeof (struct inst_name_struct));
 rec_num=get_inst_location(inst_name);
 list=list->next;
 list->position=rec_num;
 strcpy(list->instr,inst_name);
 list->next=NULL;
 return(list->position);
};
```

Appendix K

CONVERSION OF INSTRUCTION DATABASE PROGRAM

```
/*
  CON4

  This program converts an instruction file into a direct access file named
  'instructions.' This file is used by 'print-instruction',
  a function written by M. Erdly. The function was modified in 10/88 to
  generate a direct access file. The input file has the following format:

  Instruction Name: instr_name1
   "instructions..." $N "instruction..." $2 ...
  Instruction Name: instr_name2
   "instructions..." $3 ...
        ...

  $N denotes variable number N. ($2 and $3 are just examples above). Anything
  else in the file is ignored. The output list has the following form:

 instr_name1 "instructions..." N "instructions..." 2 ..
--512 byte boundry--
 instr_name2 "instructions..." 3 ...
--512 byte boundry--

  For a description of the meaning of N, see the external documentation.

  CON4 does recognize and delete the following line:

                        {",$3,"}"

  Note that anything in the input file that does not appear within quotes is
  ignored, except for variable strings (identified by '$') and the instruction
  name identifier ('Instruction Name: instr_name').
*/

#include <stdio.h>
#define TRUE 1
#define FALSE 0
#define MAX_LENGTH 200     /* maximum length of input string */
#define ER_MSG "***Error on this line***\n"
#define RECSIZE 512

FILE *fp1, *fp2;
```

```
main()
{
  char string[MAX_LENGTH];  /* array into which input line is placed */
  char infile[30], outfile[30];  /* input, output files */
  char c[6];
  int PP, FIRST, mode, a, tab, n, length, num, k, k1, k3;
  long int numchar; /* to handle files longer than 32k bytes */
  do {  /* Read in the input file name */
    printf("Enter the name of the input file... ");
    scanf("%s",infile);
    if ((fp1=fopen(infile,"r")) == NULL)
      printf("No such file exists.\n");
  } while (fp1 == NULL) ;

  printf("Do you wish the output to be sent to the screen (s) or a file (f) ");
  scanf("%s",c);
  if (c[0] == 'f')
    do {  /* Read in the output file name */
      printf("Enter the name of the output file... ");
      scanf("%s",outfile);
      if ((a=access(outfile,0)) == 0) {
        printf("The file already exists.\n");
        printf("Do you wish to write over it (y or n)? ");
        scanf("%s",c);
      }
      if ((a==0 && (c[0]=='y' || c[0]=='Y')) || a!=0)
        fp2 = fopen(outfile,"w");
    } while (a==0 && (c[0]=='n' || c[0]=='N'));
  else fp2 = stdout;  /* stdout is the screen */
  printf("\n\n");

  numchar=0; /* the number of characters already put in file.  Used to
            do record formatting.  Instructions begin on 512 byte boundries. */
  mode = 0;
  FIRST = TRUE;
  tab = 0;
  while ((num=gets_1(string)) != 999) {
    PP = FALSE;
    if (mode==0 && strncmp("Instruction Name:",&string[num],17)==0) {
      for (k=num+17; string[k]==' ' || string[k]=='\t'; k++) ;
      for (k1=k; string[k1]!=' ' && string[k1]!='\0'; k1++) ;
      string[k1] = '\0';
      if (FIRST) {
        fprintf(fp2,"%s ",&string[k]);
        numchar+=k1-k+1;
        FIRST = FALSE;
      }
      else {
        for (k3=0; k3<RECSIZE - numchar % RECSIZE; k3++) fputc (' ',fp2);
        printf("%d characters were printed; filled at end with %d chars.\n",
              numchar, RECSIZE - numchar % RECSIZE);
        fprintf(fp2,"%s ",&string[k]);
        numchar=k1-k+1;
      }
```

```
        tab = num;
      }
      else if (strncmp("{\",$3,\"}\"",&string[num],9) == 0) mode = 0;
      else if ((length=strlen(string))<=tab || strspn(string," \t")==length) ;
      else {
        if (mode==1) {
           fputc(\"',fp2);
           numchar+=1;
         };
        for (n=tab; n<=length-1; n++)
          if (string[n] == \"') {
            if (mode == 1) mode = 0;
            else mode = 1;
            fputc(\"',fp2);
             numchar+=1;
            PP = TRUE;
          }
          else if (mode==0 && string[n]=='$') {
             fprintf(fp2," %c ",string[n+1]);
              numchar+=3;
             PP = TRUE;
          }
          else if (mode == 1) {
             fputc(string[n],fp2);
              numchar+=1;
             PP = TRUE;
          }
      }
      if (PP && mode==1) {
         numchar+=4;
         if (numchar > RECSIZE) {
          fprintf(fp2,"\" -1 ");
          numchar=numchar-RECSIZE+1;
         }
         else fprintf(fp2,"\" 0 ");
      }
      else if (PP) {
         fprintf(fp2," ");
         numchar+=1;
      };
    }
    close(fp2);
    close(fp1);
}

gets_1(string)
char *string;
{
    int k;
    if (fgets(string,MAX_LENGTH,fp1) == NULL) return(999);
    for (k=0; string[k]==' ' || string[k]=='\t'; k++) ;
    string[strcspn(string,"\n")] = '\0';
    return(k);
}
```

Appendix L

CONVERSION PROGRAM TO RESTORE DATABASE FORMAT

```
/*

*/

/*declarations:*/

#include <stdio.h>;
#include <sys/file.h>;
#include "string.h"
#define LINE_LENGTH  145

struct lines
{
 char info[LINE_LENGTH];
} data[10000];

char response[30], answer, answer1[5], *temp, module[30], cause[50],
    effect[50], precondition[30], name[30], test_point[30], parameter[30],
    units[20], okreadin[10], okreadin2[10], okreadin3[10],
     min[10], max[10], lilnum[10];
int  ch, ch1, i, ii, filelength, counter2, length, space, numone,
    column[LINE_LENGTH], col, storenum, nextone, place;

FILE *fopen(), *fp1, *fp2, *fp3;

main()
{
/*open input file*/
 do
 {
  printf("What is the name of your rule file?  ");
  scanf("%s", response);
  if ((fp1=fopen(response, "r")) == NULL)
    printf("File doesn't exist.\n");
 } while (fp1 == NULL);

 do
 {
  printf("What is the name of your test file?  ");
  scanf("%s", response);
  if ((fp3=fopen(response, "r")) == NULL)
    printf("File doesn't exist.\n");
 } while (fp3 == NULL);
```

```
/*open output file*/
 do
 {
  printf("What is the name of the output file? ");
  scanf("%s", response);
  if (ch=access(response, 0) == 0)
   {
    printf("The file already exists. Overwrite Y/N? ");
    scanf("%s", answer1);
   }
  if ((ch==1 && (answer1[0]=='y' || answer1[0]=='Y')) || ch != 1)
   fp2=fopen(response, "w");
 } while (ch == 1 && (answer1[0]=='N' || answer1[0]=='n'));

/*write & read*/
 filelength=0;
 numone=0;
 while (answer=fgets(data[filelength].info, LINE_LENGTH, fp1) != NULL)
   ++filelength;
 fprintf(fp2, "NIL\nNIL\nNIL\n(");
 i=0;          /*filelength is how long file is*/
 do
 {
  if ((temp=strpbrk(data[i].info, "W")) != NULL)
    if (strncmp (temp, "WORKING", 7) == 0)  /*skip 2 lines if W is found*/
      i=i+2;
  if (((temp=strpbrk(data[i].info, "M")) != NULL) &&  (strncmp (temp, "Modu", 4) ==0))
              /*look for the letter M*/
   {
      if (numone != 0)
      {
        numone=0:  /*print control*/
       fprintf(fp2," )))\n");
      }
     for (counter2=0; counter2<=30; ++counter2)
       module[counter2]='\0';
     for (ii=8; temp[ii] != '\n'; ++ ii)
       module[ii-8]=temp[ii];
     i=i+3;
     fprintf(fp2, " ((NAME %s) (FRATE 1)\n     (CAUSAL-RULES\n", module);
   }
    else
   {
     ii=0;
    place=3;
    if ((length=strlen(data[i].info)) > 15)
     if (((strpbrk(data[i].info, "[")==NULL) && (data[i].info[ii]!='N')) ||
        (((temp=strpbrk(data[i].info, "["))!=NULL) && (strlen(data[i].info)-strlen(temp)>5)))
     {          /*is the line [deleted] ? */
       ii=85;
      for (counter2=85; data[i].info[counter2] != '\n'; ++counter2)
        if (data[i].info[counter2] != ' ')
         precondition[counter2-ii]=data[i].info[counter2];
          else
```

```
        ++ii;
      space=0;
      for (counter2=3; counter2<=length && space != 5; ++counter2)
       if ((cause[counter2-3]=data[i].info[counter2]) == ' ')
          ++space;
      space=0;
      for (counter2=40; counter2<=length && space != 4; ++counter2)
       if ((effect[counter2-40]=data[i].info[counter2]) == ' ')
          ++space;
     if (numone == 0)
      {
        ++numone;
        fprintf(fp2, "        (\n");
      }
          /*find if the cause is the same as the module*/
       if (strlen(strpbrk(cause, " ")) < 5)
         fprintf(fp2,"        (%s %s\n             (%s))\n",precondition,cause, effect);
        else
         fprintf(fp2,"        (%s (%s)\n           (%s))\n",precondition, cause, effect);
    }
   }
     /*clean up the variables*/
  for (counter2=0; counter2<=50; ++counter2)
  {
   cause[counter2]='\0';
   effect[counter2]='\0';
  }
  for (counter2=0; counter2<=30; ++counter2)
   precondition[counter2]='\0';
   ++i;
 } while (i<=filelength);
 fprintf(fp2," )))\n)\n");

/*rules database*/
 filelength=0;
 while (answer=fgets(data[filelength].info, LINE_LENGTH, fp3) != NULL)
   ++filelength;
 fprintf(fp2, "( ");
 i=0;
 ch=4;
 if (ch<filelength)
 find_col(data[ch].info, column); /*sets up the columns into int array column*/
 do
 {
  if ((strpbrk(data[i].info, "*")) != NULL) /*skip over any line with **'s */
    ++i;
   else
   {
    ii=0;
   answer=data[i].info[ii];
   if (((length=strlen(data[i].info)) > 55) && (answer != ' '))
   {       /*does it begin with a character?*/
     col=1;  /*this is the column number*/
     storenum= -1;
```

```
    for (counter2=0; counter2<=length && col < 6; counter2++)
     if (data[i].info[counter2] != ' ' ||
        (data[i].info[counter2] == ' ' && column[counter2] != 0))
        switch(col)
        {       /*set up variables depending on column*/
         case 1:  if (storenum == -1) storenum=counter2;
                 name[counter2-storenum]=data[i].info[counter2];
                   nextone=1;
                   break;
         case 2:  if (storenum == -1) storenum=counter2;
                  test_point[counter2-storenum]=data[i].info[counter2];
                   nextone=1;
                   break;
         case 3:  if (storenum == -1) storenum=counter2;
                 parameter[counter2-storenum]=data[i].info[counter2];
                   nextone=1;
                   break;
         case 4:  if (storenum == -1) storenum=counter2;
                  units[counter2-storenum]=data[i].info[counter2];
                   nextone=1;
                   break;
         case 5:  if (storenum == -1) storenum=counter2;
                 okreadin[counter2-storenum]=data[i].info[counter2];
                   nextone=1;
                   break;
        }
        else
        if (nextone == 1)
        {
         ++col; /*increment the column number*/
         nextone=0; /*skip over spaces between columns without
                      incrementing col */
         storenum= -1; /*the starting place for indivdual variables*/
        }


/*****************************************************************
*      hi lo ok value (ok (min max) lo (-inf min) hi (max inf))
*      hi lo ok noval (ok lo hi)
*      ok bad   value (ok (min max) bad (-inf min) (max inf))
*      ok bad   noval (ok bad)
*****************************************************************/

/*the following method is used instead of the above column method because
  sometimes the data doesn't exist and the program looks for nothing or
  misinterprets data found*/
      ch=63;
      if (data[i].info[ch] != ' ') /*there's s.t. there!!!*/
      {
       for (counter2=63; data[i].info[counter2] != ' '; ++counter2)
        min[counter2-63]=data[i].info[counter2];
        if (strlen (min) < 6)
         for (counter2=69; data[i].info[counter2] != ' ' ||   counter2<70; ++counter2)
          max[counter2-69]=data[i].info[counter2];
```

```
        else
        {
         ch1=67;
        if (data[i+1].info[ch1] != '\0')
        for (counter2=67; data[i+1].info[counter2] != ' ' || counter2<70; ++counter2);
        }
         max[counter2-67]=data[i+1].info[counter2];
        /*the data[i+1] is used above because if the qual val is to long,
          there's no space for the second on the same line, and it's
          located underneath*/
      }
       else
       if (data[i].info[ch+1] != ' ')
        {
        for (counter2=64; data[i].info[counter2] != ' '; ++counter2)
         min[counter2-64]=data[i].info[counter2];
         if (strlen (min) < 6)
          for (counter2=70; data[i].info[counter2] != ' ' ||       counter2<71; ++counter2)
           max[counter2-70]=data[i].info[counter2];
          else
          {
           ch1=68;
          if (data[i+1].info[ch1] != '\0')
          for (counter2=68; data[i+1].info[counter2] != ' ' ||
                      counter2<71; ++counter2)
           max[counter2-68]=data[i+1].info[counter2];
          }
       }
      if (atof(min) > atof(max))
        printf("Problem with min max values in %s %s.\n", name, test_point);
             /*if the min is greater than the max, there's a problem.*/
      ii=58; /*this is done because sometimes the data's in the wrong space*/
      for (counter2=58; counter2<60; ++counter2)
       if (data[i+1].info[counter2] != ' ')
        okreadin2[counter2-ii]=data[i+1].info[counter2];
         else
          ++ii;
      if (strlen(data[i+2].info) >58)
       for (counter2=58; data[i+2].info[counter2] != ' ' &&
                    counter2<strlen(data[i+2].info); ++counter2)
        okreadin3[counter2-58]=data[i+2].info[counter2];
       ii=77;
      for (counter2=77; counter2<80; ++counter2)
       if (data[i].info[counter2] != ' ')
        lilnum[counter2-ii]=data[i].info[counter2];
        else
         ++ii;

/******** printing time *********/
      fprintf(fp2, "   (%s (%s %s\n          S1\n",
              name, test_point, parameter);
     if ((okreadin[0] == 'f') && (okreadin2[0] == 'o'))
       fprintf(fp2, "          ((ok) (faulted))\n");
     if ((okreadin[0] == 'a') && (okreadin2[0] == 'o'))
```

```
      fprintf(fp2, "          ((absent) (ok))\n");
    if ((okreadin[0] == 'p') && (okreadin2[0] == 'o'))
      fprintf(fp2, "          ((ok) (present))\n");
    if ((okreadin[0] == 'o') && (okreadin2[0] == 'o'))
      fprintf(fp2, "          ((ok) (on))\n");
    if ((okreadin[0] == 'b') && (okreadin2[0] == 'o'))
      if (min[0] == '\0')
       fprintf(fp2, "          ((ok) (bad))\n");
       else
       fprintf(fp2,"          ((ok (%s %s)) (bad (-inf %s) (%s inf)))\n",
                  min, max, min, max);
    if ((okreadin[0] == 'h') && (okreadin2[0] == 'l'))
      if (min[0] == '\0')
       fprintf(fp2, "          ((ok) (lo) (hi))\n");
       else
        fprintf(fp2,
         "          ((ok ((%s %s))) (lo ((-inf %s))) (hi ((%s inf))))\n",
            min, max, min, max);

     fprintf(fp2, "        %s\n        D\n", units);
     fprintf(fp2, "        %s\n        NIL))\n", lilnum);
    }
  }
  for (counter2=0; counter2<=30; ++counter2)
  {
   name[counter2]='\0';
   test_point[counter2]='\0'; /*cleaning up the variables*/
   parameter[counter2]='\0';
  }
  for (counter2=0; counter2<=20; ++counter2)
    units[counter2]='\0';
  for (counter2=0; counter2<=10; ++counter2)
  {
   okreadin[counter2]='\0';
   okreadin2[counter2]='\0';
   okreadin3[counter2]='\0';
    lilnum[counter2]='\0';
    min[counter2]='\0';
    max[counter2]='\0';
  }
   ++i;
 } while (i<=filelength);
 fprintf(fp2,")\n");
 fprintf(fp2,"NIL\nNIL\nNIL\n");
 close(fp1);
 close(fp3);
 close(fp2);
}

/*************************** FUNCTIONS ****************************/

find_col (string, columns)

char *string;
```

```
int columns[];

{
 int k=0, num=0;

 while (string[k] == ' ') columns[k++]=1;
 if (k==0) columns[k++]=1;
 while (string[k] != '\0')
 {
  if (string[k]==' ') columns[k]=0;
    else
     if (string[k-1] != ' ') columns[k] = num;
      else
       columns[k] = ++num;
   k++;
 }
 while (k < LINE_LENGTH) columns[k++]=0;
 return;
}

/*********************************************************************
* This function looks at a header and develops column fields:
*
*   Header1        H2    Head3            Header4
* 11111111110002200333330000000444444400000
*
* as such, to find out what column a specific piece of data
* is in.
*********************************************************************/
```

Appendix M

SAMPLE DATA OUTPUT FORMAT FROM RULE VERIFIER

NODE	PARENT	PRE	CHILD	PRE
(A10J1 CIRCULATE BAD)	(A26-35 CIRCULATE BAD)	T	(A10J9 RE1 BAD) A1A10J1	T
(A10J2 LOGIC_LEVELS BAD)	(A77J4 VOLTS BAD)	T	(A10J2 RE2 BAD) A10_DELAY_LINE	T T
(A10J2 RE2 BAD)	(A26-15 RE2 BAD) (A10J2 LOGIC_LEVELS BAD)	T T	(A10J9 RE1 BAD) A1A10J2	T
(A10J3 END_CLEAR BAD)	(A26A1S8-3 END_CLEAR BAD)	END_CLEAR_USED	(A10J9 RE1 BAD) A1A10J3	T
(A10J4 LOAD_END_REF BAD)	(A26-5 LOAD_END_REF BAD) (A10J4 LOGIC_LEVELS BAD)	T T	(A10J9 RE1 BAD) A1A10J4	T
(A10J4 LOGIC_LEVELS BAD)	(A77J4 VOLTS BAD) A10_DELAY_LINE	T T	(A10J4 LOAD_END_REF BAD)	T
(A10J5 LOAD_END_ADVANCE BAD)	(A26-32 LOAD_END_ADVANCE BAD)	T	(A10J9 RE1 BAD) A1A10J5	T
(A10J6 ST1 BAD)	(A10-9 ST1 BAD)	T	(A10J9 RE1 BAD) A1A10J6	T
(A10J7 NOT_CL1 BAD)	(A10-21 NOT_CL1 BAD)	T	(A10J9 RE1 BAD) A1A10J7	T
(A10J8 DST1 BAD)	(A29-9 DST1 BAD)	T	(A10J9 RE1 BAD) A1A10J8	T
(A10J9 LOGIC_LEVELS BAD)	(A77J4 VOLTS BAD) A10_DELAY_LINE	T T	(A10J9 RE1 BAD)	T
(A10J9 LOGIC_LEVELS BAD)				

(A10J9 RE1 BAD)	(A10J1 CIRCULATE BAD) T	(A26-13 RE1 BAD)	T
	(A10J5 LOAD_END_ADVANCE BAD) T	(A10-10 NOT_RE1 BAD)	T
	(A10J4 LOAD_END_REF BAD) T	A1A10J9	
	(A10J8 DST1 BAD) T		
	(A10J6 ST1 BAD) T		
	(A10J7 NOT_CL1 BAD) T		
	(A10J3 END_CLEAR BAD) T		
	(A77J4 VOLTS BAD) T		
	(A76J3 VOLTS BAD) T		
	A10_DELAY_LINE T		
	(A10J2 RE2 BAD) T		
	(A10J9 LOGIC_LEVELS BAD) T		
(A11J10 ATOD_CLEAR BAD)	(A14-23 ATOD_CLEAR BAD) T	(A23 AMPLITUDE BAD)	T
	(A18-22 ATOD_CLEAR_AMPL BAD) T	AD_CLEAR	
(A11J11 ATOD_SET_SIGN BAD)	(A14-24 ATOD_SET_SIGN BAD) T	(A23-3 INVERTED_SIGN BAD)	T
		(A23-6 SIGN BAD)	T
		AD_SET_SIGN	
(A11J12 ATOD_CLOCK BAD)	(A14-7 ATOD_CLOCK BAD) T	(A23 AMPLITUDE BAD)	T
		AD_CLOCK	
(A11J14 GATE_4 BAD)	(A15 FUNCTIONS BAD) T	(A15-38 SIGNAL FAULTED)	T
		(A61J7 GATE_SELECT BAD)	T
		(A22 GATES_OR_PWR_OR_MODULE BAD) T	
		A1A11J14	
(A11J15 NOT_LOAD_REF BAD)	(A15-14 NOT_LOAD_REF BAD) T	(A26-5 LOAD_END_REF BAD) T	
		A1A11J15	
(A11J15 NOT_LOAD_REF BAD)			
(A11J16 STORE_LEFT_BEAM BAD)	(A15-3 STORE_LEFT_BEAM BAD) T	(A7J4 SIGN_LEFT_BEAMS BAD) T	
		(A17-OUT AMPL_LEFT_BEAMS BAD) T	
		ST_LEFT_BM	
(A11J18 SYNC BAD)	(A15-2 SYNC BAD) T	(A26E3-Q END_SYNC BAD) T	
		A1A11J18	
(A11J2 CL1 BAD)	(A14-10 CL1 BAD) T	(A7-20 CL1 BAD)	T
		(A10-20 CL1 BAD)	T
		A1A11J2	
(A11J3 CL2 BAD)	(A14-35 CL2 BAD) T	(A4-20 CL2 BAD)	T
		(A2-22 NOT_CL2 BAD)	T
		A1A11J3	

(A11J4 ST3 BAD)	(A14 FUNCTIONS BAD)	T	(A15 FUNCTIONS BAD) T (A18 CONTROL_FUNCTIONS BAD) T A1A11J4
(A11J5 NOT_ST1 BAD)	(A14-4 NOT_ST1 BAD)	T	(A7-8 NOT_ST1 BAD) T (A10-8 NOT_ST1 BAD) T A1A11J5
(A11J6 NOT_ST2 BAD)	(A14-5 NOT_ST2 BAD)	T	(A4-9 ST2 BAD) T (A1-9 ST3 BAD) T A1A11J6
(A11J7 WAVEFORM BAD)	(A14-15 G1 BAD)	T	(A58J7 GATE_SELECT BAD) T (A59J7 GATE_SELECT BAD) T (A60J7 GATE_SELECT BAD) T
(A11J7 WAVEFORM BAD)			(A61J7 GATE_SELECT BAD) T A1A11J7
(A11J8 G2 BAD)	(A14 FUNCTIONS BAD)	T	(A15 FUNCTIONS BAD) T (A18 CONTROL_FUNCTIONS BAD) T (A27J4 G2 BAD) T A1A11J8
(A11J9 WAVEFORM BAD)	(A14 FUNCTIONS BAD)	T	A1A11J9
(A13J1 FREQUENCY HI)	(A13J1 WAVEFORM BAD)	T	A1A13J1_F
(A13J1 FREQUENCY LO)	(A13J1 WAVEFORM BAD)	T	A1A13J1_F
(A13J1 LOGIC_LEVELS BAD)	(A13J1 WAVEFORM BAD)	T	A1A13J1_L
(A13J1 WAVEFORM BAD)	TIMING (A77J4 VOLTS BAD) (A77J7 VOLTS BAD) (A76J3 VOLTS BAD)	T T T T	(A13J1 LOGIC_LEVELS BAD) T (A13J1 FREQUENCY HI) T (A13J1 FREQUENCY LO) T (A14 FUNCTIONS BAD) T (A19J7 SIGNAL FAULTED) T
(A1J1 NOT_MTS_GATE BAD)	(A15-5 NOT_MTS_GATE BAD)	T	(A1-13 BIT_0_LEFT_BEAMS BAD) T A1A1J1
(A1J3 END_CLEAR BAD)	(A26A1S8-3 END_CLEAR BAD) END_CLEAR_USED		(A1-13 BIT_0_LEFT_BEAMS BAD) END_CLEAR_USED A1A1J3

(A1J4 BIT_0_LEFT_INPUT BAD) (A22_A17-OUT BEAM_1_WAVE BAD) T (A1-13 BIT_0_LEFT_BEAMS BAD) T
(A22_A17-OUT BEAM_2_WAVE BAD) T
(A22_A17-OUT BEAM_3_WAVE BAD) T

(A1J4 BIT_0_LEFT_INPUT BAD) (A22_A17-OUT BEAM_4_WAVE BAD) T (A1J4 LOGIC_LEVELS BAD) T

(A1J4 LOGIC_LEVELS BAD) (A77J4 VOLTS BAD) T (A1J4 BIT_0_LEFT_INPUT BAD) T
A1_DELAY_LINE T A1A1J4

(A1J5 MTS_GATE BAD) (A15-4 MTS_GATE BAD) T (A1-13 BIT_0_LEFT_BEAMS BAD) T
A1A1J5

(A1J6 ST3 BAD) (A1-9 ST3 BAD) T (A1-13 BIT_0_LEFT_BEAMS BAD) T
A1A1J6

(A1J7 NOT_CL2 BAD) (A2-22 NOT_CL2 BAD) T (A1-13 BIT_0_LEFT_BEAMS BAD) T
A1A1J7

(A1J8 DST3 BAD) (A3-9 DST3 BAD) T (A1-13 BIT_0_LEFT_BEAMS BAD) T
A1A1J8

(A1J9 LOGIC_LEVELS BAD) (A77J4 VOLTS BAD) T (A1-13 BIT_0_LEFT_BEAMS BAD) T
A1_DELAY_LINE T A1A1J9

(A20J11 NOT_PC BAD) (A18 CONTROL_FUNCTIONS BAD) T (A27-34 SIGNAL FAULTED) T
(A20J12 PC BAD) T
A1A20J11

(A20J13 SIGNAL FAULTED) (A17 AMPL_CORREL_LEFT_BEAMS BAD) T (A27-15 SIGNAL FAULTED) T
(A18 LEFT_TEST BAD) T (A27-34 SIGNAL FAULTED) T
(A21J5 SIGNAL FAULTED) T A1A20J13

(A20J2 GATE_1 BAD) (A15-13 NOT_GATE_1 BAD)] T (A30 FUNCTIONS BAD) T
TIMING T A1A20J2

(A20J2 GATE_1 BAD)

(A20J3 GATE_2 BAD) (A15-15 NOT_GATE_2 BAD) T (A31 FUNCTIONS BAD) T
TIMING T A1A20J3

(A20J4 GATE_3 BAD) (A15-6 NOT_GATE_3 BAD) T (A32 FUNCTIONS BAD) T
TIMING T A1A20J4

(A20J5 GATE_4 BAD) (A15-7 NOT_GATE_4 BAD) T (A33 FUNCTIONS BAD) T
TIMING T A1A20J5

(A20J6 ATOD_CLEAR_SIGN BAD) (A18-25 ATOD_CLEAR_SIGN BAD) T(A23-3 INVERTED_SIGN BAD) T
(A23-6 SIGN BAD) T
AD_CLR_SIGN

(A20J8 ATOD_SET BAD) (A18-45 ATOD_SET BAD) T (A23-3 INVERTED_SIGN BAD) T
(A23-6 SIGN BAD) T
(A23 AMPLITUDE BAD) T
AD_SET

(A20J9 NOT_PE BAD) (A18 CONTROL_FUNCTIONS BAD) T (A20J10 PE BAD) T
A1A20J9

(A21J11 END_SYNC BAD) (A26-18 END_SYNC BAD) T (A18 CONTROL_FUNCTIONS BAD) T
A1A21J11

(A21J13 RE3 BAD) (A26-29 RE3 BAD) T (A17 AMPL_CORREL_LEFT_BEAMS
BAD) T

A1A21J13

(A21J14 NOT_RE3 BAD) (A26-30 NOT_RE3 BAD) T (A17 AMPL_CORREL_LEFT_BEAMS
BAD) T

A1A21J14

(A21J5 LEFT_TEST_NOT_SIGN BAD) (A22-3 LEFT_TEST_NOT_SIGN BAD) T LFT_TST_SGN

(A21J5 SIGNAL FAULTED) (A22-3 LEFT_TEST_NOT_SIGN BAD) T (A20J13 SIGNAL FAULTED) T

(A21J8 REF BAD) (A26-9 REF BAD) T A1A21J8

(A24J1 TEST_VOLTS BAD) (A26A1S10-2 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1
TIME_SLOT_1_VOLTS HI) A26A1S9_NOT_SET_TO_OPER
(A26A1S10-2 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1
TIME_SLOT_4_VOLTS HI) A26A1S9_NOT_SET_TO_OPER
(A24-10 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1
TIME_SLOT_7_VOLTS HI) A26A1S9_NOT_SET_TO_OPER
(A24-8 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1
TIME_SLOT_10_VOLTS HI) A26A1S9_NOT_SET_TO_OPER
(A24-6 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1
TIME_SLOT_1_VOLTS LO) A26A1S9_NOT_SET_TO_OPER
(A24-13 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1
TIME_SLOT_4_VOLTS LO) A26A1S9_NOT_SET_TO_OPER
(A24-10 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1
TIME_SLOT_7_VOLTS LO) A26A1S9_NOT_SET_TO_OPER
(A24-8 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1
TIME_SLOT_10_VOLTS LO) A26A1S9_NOT_SET_TO_OPER
(A24-6 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER
(A24-13 VOLTS BAD) A26A1S9_NOT_SET_TO_OPER

(A24J1 TIME_SLOT_10_VOLTS HI) (A26A1S9-1 TIME_SLOT_10_VOLTS HI) A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_10_VOLTS HI) T
(A24J1 TEST_VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1 VOLTS HI) T
(A24J2 TIME_SLOT_10_VOLTS HI) T
(A24J1 VOLTS HI) T
MUX_AMPJ1_10

(A24J1 TIME_SLOT_10_VOLTS LO) (A26A1S9-1 TIME_SLOT_10_VOLTS LO) A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_10_VOLTS LO) T
A26A1S9 A26A1S9_SET_TO_OPER (A24J1 VOLTS LO) T
(A24J1 TEST_VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J2 TIME_SLOT_10_VOLTS LO) T
(A24J1 VOLTS LO) T
MUX_AMPJ1_10

(A24J1 TIME_SLOT_10_WAVEFORM BAD) (A26A1S9-1 TIME_SLOT_10_WAVEFORM BAD) A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_10_WAVEFORM BAD) T
(A24J2 TIME_SLOT_10_WAVEFORM BAD) T

(A24J1 TIME_SLOT_10_WAVEFORM BAD)

(A24J1 TIME_SLOT_1_VOLTS HI) (A26A1S9-1 TIME_SLOT_1_VOLTS HI) A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_1_VOLTS HI) T
(A24J1 TEST_VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1 VOLTS HI) T
(A24J2 TIME_SLOT_1_VOLTS HI) T
(A24J1 VOLTS HI) T
MUX_AMPJ1_1

(A24J1 TIME_SLOT_1_VOLTS LO) (A26A1S9-1 TIME_SLOT_1_VOLTS LO) A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_1_VOLTS LO) T
A26A1S A26A1S9_SET_TO_OPER (A24J1 VOLTS LO) T
(A24J1 TEST_VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J2 TIME_SLOT_1_VOLTS LO) T
(A24J1 VOLTS LO) T
MUX_AMPJ1_1

(A24J1 TIME_SLOT_1_WAVEFORM BAD) (A26A1S9-1 TIME_SLOT_1_WAVEFORM BAD) A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_1_WAVEFORM BAD) T
(A24J2 TIME_SLOT_1_WAVEFORM BAD) T

(A24J1 TIME_SLOT_4_VOLTS HI) (A26A1S9-1 TIME_SLOT_4_VOLTS HI) A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_4_VOLTS HI) T
(A24J1 TEST_VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1 VOLTS HI) T
(A24J2 TIME_SLOT_4_VOLTS HI) T
(A24J1 VOLTS HI) T
MUX_AMPJ1_4

```
-------------------------------------------------------------------------------
(A24J1 TIME_SLOT_4_VOLTS LO) (A26A1S9-1 TIME_SLOT_4_VOLTS LO) A26A1S9_SET_TO_OPER
(A24J2 TIME_SLOT_4_VOLTS LO) T
                    A26A1S9                         A26A1S9_SET_TO_OPER (A24J1
VOLTS LO) T
                    (A24J1 TEST_VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J2
TIME_SLOT_4_VOLTS LO) T
                                                    (A24J1 VOLTS LO)   T
                                                    MUX_AMPJ1_4

-------------------------------------------------------------------------------
(A24J1 TIME_SLOT_4_WAVEFORM BAD) (A26A1S9-1 TIME_SLOT_4_WAVEFORM BAD)
A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_4_WAVEFORM BAD) T
                                                    (A24J2 TIME_SLOT_4_WAVEFORM
BAD) T

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
(A24J1 TIME_SLOT_7_VOLTS HI) (A26A1S9-1 TIME_SLOT_7_VOLTS HI) A26A1S9_SET_TO_OPER
(A24J2 TIME_SLOT_7_VOLTS HI) T
                    (A24J1 TEST_VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J1 VOLTS HI)
T
                                                    (A24J2 TIME_SLOT_7_VOLTS HI) T
                                                    (A24J1 VOLTS HI)   T
                                                    MUX_AMPJ1_7

-------------------------------------------------------------------------------
(A24J1 TIME_SLOT_7_VOLTS LO) (A26A1S9-1 TIME_SLOT_7_VOLTS LO) A26A1S9_SET_TO_OPER
(A24J2 TIME_SLOT_7_VOLTS LO) T
                    A26A1S9                 A26A1S9_SET_TO_OPER (A24J1 VOLTS LO) T
                    (A24J1 TEST_VOLTS BAD) A26A1S9_NOT_SET_TO_OPER (A24J2
TIME_SLOT_7_VOLTS LO) T
                                                    (A24J1 VOLTS LO)   T
                                                    MUX_AMPJ1_7

-------------------------------------------------------------------------------
(A24J1 TIME_SLOT_7_WAVEFORM BAD) (A26A1S9-1 TIME_SLOT_7_WAVEFORM BAD)
A26A1S9_SET_TO_OPER (A24J2 TIME_SLOT_7_WAVEFORM BAD) T
                                            (A24J2 TIME_SLOT_7_WAVEFORM BAD) T

-------------------------------------------------------------------------------
(A24J1 VOLTS HI)    (A24J1 TIME_SLOT_1_VOLTS HI) T   MUX_AMPJ1_AV
                    (A24J1 TIME_SLOT_4_VOLTS HI) T
                    (A24J1 TIME_SLOT_7_VOLTS HI) T
                    (A24J1 TIME_SLOT_10_VOLTS HI) T
                    (A24J1 TIME_SLOT_1_VOLTS HI) T
                    (A24J1 TIME_SLOT_4_VOLTS HI) T
                    (A24J1 TIME_SLOT_7_VOLTS HI) T
                    (A24J1 TIME_SLOT_10_VOLTS HI) T

-------------------------------------------------------------------------------
(A24J1 VOLTS LO)    (A24J1 TIME_SLOT_1_VOLTS LO) T   MUX_AMPJ1_AV
                    (A24J1 TIME_SLOT_4_VOLTS LO) T
                    (A24J1 TIME_SLOT_7_VOLTS LO) T
                    (A24J1 TIME_SLOT_10_VOLTS LO) T
                    (A24J1 TIME_SLOT_1_VOLTS LO) T
                    (A24J1 TIME_SLOT_4_VOLTS LO) T
                    (A24J1 TIME_SLOT_7_VOLTS LO) T
```

(A24J1 VOLTS LO) (A24J1 TIME_SLOT_10_VOLTS LO) T

(A24J2 TIME_SLOT_10_VOLTS HI) (A24J1 TIME_SLOT_10_VOLTS HI) T (A24J2 VOLTS HI) T
(A24 FUNCTIONS BAD) T (A24J2 VOLTS HI) T
(A24J1 TIME_SLOT_10_VOLTS HI) T (A23_OUTPUT TIME_SLOT_10_AMPL HI) T
(A24 FUNCTIONS BAD) T MUX_AMPJ2_10

(A24J2 TIME_SLOT_10_VOLTS LO) (A24J1 TIME_SLOT_10_VOLTS LO) T (A24J2 VOLTS LO) T
(A24 FUNCTIONS BAD) T (A24J2 VOLTS LO) T
(A24J1 TIME_SLOT_10_VOLTS LO) T (A23_OUTPUT TIME_SLOT_10_AMPL LO) T
(A24 FUNCTIONS BAD) T MUX_AMPJ2_10

(A24J2 TIME_SLOT_10_WAVEFORM BAD) (A24J1 TIME_SLOT_10_WAVEFORM BAD) T (A23_OUTPUT TIME_SLOT_10_WAVE BAD) T
(A24 FUNCTIONS BAD) T
(A24J1 TIME_SLOT_10_WAVEFORM BAD) T
(A24 FUNCTIONS BAD) T

(A24J2 TIME_SLOT_1_VOLTS HI) (A24J1 TIME_SLOT_1_VOLTS HI) T (A24J2 VOLTS HI) T
(A24 FUNCTIONS BAD) T (A24J2 VOLTS HI) T
(A24J1 TIME_SLOT_1_VOLTS HI) T (A23_OUTPUT TIME_SLOT_1_AMPL HI) T
(A24 FUNCTIONS BAD) T MUX_AMPJ2_1

(A24J2 TIME_SLOT_1_VOLTS LO) (A24J1 TIME_SLOT_1_VOLTS LO) T (A24J2 VOLTS LO) T
(A24 FUNCTIONS BAD) T (A24J2 VOLTS LO) T
(A24J1 TIME_SLOT_1_VOLTS LO) T (A23_OUTPUT TIME_SLOT_1_AMPL LO) T
(A24 FUNCTIONS BAD) T MUX_AMPJ2_1

(A24J2 TIME_SLOT_1_WAVEFORM BAD) (A24J1 TIME_SLOT_1_WAVEFORM BAD) T (A23_OUTPUT TIME_SLOT_1_WAVE BAD) T
(A24 FUNCTIONS BAD) T
(A24J1 TIME_SLOT_1_WAVEFORM BAD) T
(A24 FUNCTIONS BAD) T

(A24J2 TIME_SLOT_1_WAVEFORM BAD)

(A24J2 TIME_SLOT_4_VOLTS HI) (A24J1 TIME_SLOT_4_VOLTS HI) T (A24J2 VOLTS HI) T
(A24 FUNCTIONS BAD) T (A24J2 VOLTS HI) T
(A24J1 TIME_SLOT_4_VOLTS HI) T (A23_OUTPUT TIME_SLOT_4_AMPL HI) T
(A24 FUNCTIONS BAD) T MUX_AMPJ2_4

(A24J2 TIME_SLOT_4_VOLTS LO) (A24J1 TIME_SLOT_4_VOLTS LO) T (A24J2 VOLTS LO) T
(A24 FUNCTIONS BAD) T (A24J2 VOLTS LO) T
(A24J1 TIME_SLOT_4_VOLTS LO) T (A23_OUTPUT TIME_SLOT_4_AMPL LO) T
(A24 FUNCTIONS BAD) T MUX_AMPJ2_4

(A24J2 TIME_SLOT_4_WAVEFORM BAD) (A24J1 TIME_SLOT_4_WAVEFORM BAD) T (A23_OUTPUT TIME_SLOT_4_WAVE BAD) T
(A24 FUNCTIONS BAD) T
(A24J1 TIME_SLOT_4_WAVEFORM BAD) T
(A24 FUNCTIONS BAD) T

(A24J2 TIME_SLOT_7_VOLTS HI) (A24J1 TIME_SLOT_7_VOLTS HI) T (A24J2 VOLTS HI) T
(A24 FUNCTIONS BAD) T (A24J2 VOLTS HI) T
(A24J1 TIME_SLOT_7_VOLTS HI) T (A23_OUTPUT TIME_SLOT_7_AMPL HI) T
(A24 FUNCTIONS BAD) T MUX_AMPJ2_7

(A24J2 TIME_SLOT_7_VOLTS LO) (A24J1 TIME_SLOT_7_VOLTS LO) T (A24J2 VOLTS LO) T
(A24 FUNCTIONS BAD) T (A24J2 VOLTS LO) T
(A24J1 TIME_SLOT_7_VOLTS LO) T (A23_OUTPUT TIME_SLOT_7_AMPL LO) T
(A24 FUNCTIONS BAD) T MUX_AMPJ2_7

(A24J2 TIME_SLOT_7_WAVEFORM BAD) (A24J1 TIME_SLOT_7_WAVEFORM BAD) T (A23_OUTPUT TIME_SLOT_7_WAVE BAD) T
(A24 FUNCTIONS BAD) T
(A24J1 TIME_SLOT_7_WAVEFORM BAD) T
(A24 FUNCTIONS BAD) T

(A24J2 VOLTS HI) (A24J2 TIME_SLOT_1_VOLTS HI) T MUX_AMPJ2_AV
(A24J2 TIME_SLOT_4_VOLTS HI) T
(A24J2 TIME_SLOT_7_VOLTS HI) T
(A24J2 TIME_SLOT_10_VOLTS HI) T
(A24J2 TIME_SLOT_1_VOLTS HI) T
(A24J2 TIME_SLOT_4_VOLTS HI) T
(A24J2 TIME_SLOT_7_VOLTS HI) T
(A24J2 TIME_SLOT_10_VOLTS HI) T

(A24J2 VOLTS LO) (A24J2 TIME_SLOT_1_VOLTS LO) T MUX_AMPJ2_AV
(A24J2 TIME_SLOT_4_VOLTS LO) T
(A24J2 TIME_SLOT_7_VOLTS LO) T
(A24J2 TIME_SLOT_10_VOLTS LO) T
(A24J2 TIME_SLOT_1_VOLTS LO) T
(A24J2 TIME_SLOT_4_VOLTS LO) T
(A24J2 TIME_SLOT_7_VOLTS LO) T
(A24J2 TIME_SLOT_10_VOLTS LO) T

(A24J4 DELTIC_REF BAD) (A43J3 DELTIC_REF BAD) T (A24J5 DELTIC_REF BAD) T
CORRELATOR_REF T (A24J5 DELTIC_REF BAD) T
A1A24J4

(A24J5 DELTIC_REF BAD) (A24J4 DELTIC_REF BAD) T (A26-4 DELTIC_REF BAD) T
CORRELATOR_REF T A1A24J5

	(A76J3 VOLTS BAD)	T	
	(A57-21 VOLTS BAD)	T	
	(A24J4 DELTIC_REF BAD)	T	
	A24_AMP	T	
	(A76J3 VOLTS BAD)	T	
	(A57-21 VOLTS BAD)	T	

(A26A1DS8 ALL_BEAMS_ERR BAD) (A27-2 ALL_BEAMS_ERR BAD) T NIL

	PMFL	T
	(A77J4 VOLTS BAD)	T

(A26A1DS8 ALL_BEAMS_ERR ON) (A27-2 ALL_BEAMS_ERR ON) T DS8_ALL

(A26A1DS8 BEAMS_1-4_ERR BAD) (A27-2 BEAMS_1-4_ERR BAD) T NIL

	PMFL	T
	(A77J4 VOLTS BAD)	T

(A26A1DS8 BEAMS_1-4_ERR ON) (A27-2 BEAMS_1-4_ERR ON) T DS8_1-4

(A26A1DS8 LIGHT OFF)	(A27-2 OFF_POSITION BAD)	T	NIL
	PMFL	T	
	(A77J4 VOLTS BAD)	T	

(A26A1J10 DTOA_POWER HI) (A42-10 DTOA_POWER HI) T 26A1J10_SC

(A26A1J10 DTOA_POWER LO) (A42-10 DTOA_POWER LO) T 26A1J10_SC

(A26A1J9 DTOA_INHIBIT HI) (A42-5 DTOA_INHIBIT HI) T 26A1J9_SC

(A26A1J9 DTOA_INHIBIT LO) (A42-5 DTOA_INHIBIT LO) T 26A1J9_SC

(A29J6 ST1 BAD)	(A10-9 ST1 BAD)	T	A1A29J6

(A29J7 NOT_CL1 BAD)	(A10-21 NOT_CL1 BAD)	T	A1A29J7

(A29J8 DST1 BAD)	(A29-9 DST1 BAD)	T	A1A29J8

(A2J6 ST3 BAD)	(A1-9 ST3 BAD)	T	A1A2J6

(A30J1 SIGN BAD)	(A22-2 SIGN_BEAM_1 BAD)	T	(A30 FUNCTIONS BAD) A1A30J1	T

```
---------------------------------------------------------------------------------------------------
(A30J2 BIT_1 BAD)          (A17 AMPL_BEAM_1 BAD) T          (A30 FUNCTIONS BAD)      T
                                                            (A30J2 LOGIC_LEVELS BAD) T

---------------------------------------------------------------------------------------------------
(A30J2 LOGIC_LEVELS BAD) (A30J2 BIT_1 BAD)     T          A1A30J2

---------------------------------------------------------------------------------------------------
(A30J3 BIT_0 BAD)          (A17 AMPL_BEAM_1 BAD) T          (A30 FUNCTIONS BAD)      T
                                                            (A30J3 LOGIC_LEVELS BAD) T


---------------------------------------------------------------------------------------------------
(A30J3 LOGIC_LEVELS BAD) (A30J3 BIT_0 BAD)     T          A1A30J3

---------------------------------------------------------------------------------------------------
(A30J4 AMPLITUDE HI)(CORRELATOR-OUT BEAM_1_AMPL HI) T (A45J2 AMPLITUDE HI) T
                           (A42-10 DTOA_POWER HI) T      (A30J4 POS_PEAK HI)      T
                           (A30 FUNCTIONS BAD) T         (A30J4 NEG_PEAK HI)      T
                                                         (A30J4 TEST_188_VOLTS HI)
A26A1S9_NOT_SET_TO_OPER
                                                         (A30J4 TEST_212_VOLTS HI)
A26A1S9_NOT_SET_TO_OPER
                                                         (A30J4 TEST_424_VOLTS HI)
A26A1S9_NOT_SET_TO_OPER
                                                         (A30J4 TEST_636_VOLTS HI)
A26A1S9_NOT_SET_TO_OPER

---------------------------------------------------------------------------------------------------
(A30J4 AMPLITUDE LO)(CORRELATOR-OUT BEAM_1_AMPL LO) T (A45J2 AMPLITUDE LO)  T
                           (A42-10 DTOA_POWER LO) T      (A30J4 POS_PEAK LO)      T
                           (A30 FUNCTIONS BAD) T         (A30J4 NEG_PEAK LO)      T
                                                         (A30J4 TEST_188_VOLTS LO)
A26A1S9_NOT_SET_TO_OPER
                                                         (A30J4 TEST_212_VOLTS LO)
A26A1S9_NOT_SET_TO_OPER
                                                         (A30J4 TEST_424_VOLTS LO)
A26A1S9_NOT_SET_TO_OPER
                                                         (A30J4 TEST_636_VOLTS LO)
A26A1S9_NOT_SET_TO_OPER

---------------------------------------------------------------------------------------------------
(A30J4 NEG_PEAK HI)        (A30J4 AMPLITUDE HI) T           A1A30J4_NEG

---------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------
(A30J4 NEG_PEAK LO)        (A30J4 AMPLITUDE LO) T           A1A30J4_NEG

---------------------------------------------------------------------------------------------------
(A30J4 POS_PEAK HI)        (A30J4 AMPLITUDE HI) T           A1A30J4_POS

---------------------------------------------------------------------------------------------------
(A30J4 POS_PEAK LO)        (A30J4 AMPLITUDE LO) T           A1A30J4_POS

---------------------------------------------------------------------------------------------------
(A30J4 TEST_188_VOLTS HI) (A30J4 AMPLITUDE HI) A26A1S9_NOT_SET_TO_OPER AD_1_188_P

---------------------------------------------------------------------------------------------------
```

(A30J4 TEST_188_VOLTS LO) (A30J4 AMPLITUDE LO) A26A1S9_NOT_SET_TO_OPER AD_1_188_P

(A30J4 TEST_212_VOLTS HI) (A30J4 AMPLITUDE HI) A26A1S9_NOT_SET_TO_OPER AD_1_212_P

(A30J4 TEST_212_VOLTS LO) (A30J4 AMPLITUDE LO) A26A1S9_NOT_SET_TO_OPER AD_1_212_P

(A30J4 TEST_424_VOLTS HI) (A30J4 AMPLITUDE HI) A26A1S9_NOT_SET_TO_OPER AD_1_424_P

(A30J4 TEST_424_VOLTS LO) (A30J4 AMPLITUDE LO) A26A1S9_NOT_SET_TO_OPER AD_1_424_P

(A30J4 TEST_636_VOLTS HI) (A30J4 AMPLITUDE HI) A26A1S9_NOT_SET_TO_OPER AD_1_636_P

(A30J4 TEST_636_VOLTS LO) (A30J4 AMPLITUDE LO) A26A1S9_NOT_SET_TO_OPER AD_1_636_P

Appendix N

SAMPLE DATA OUTPUT FORMAT FROM AMBIGUITY SET VERIFIER

Test No.	Abnormality	Module	Immediate Effect
26A1J10_SC	HI	MOD_REFER	(A42-10 DTOA_POWER HI)
26A1J10_SC	HI	POWER	(A26_CABINET_PWR VOLTS BAD)
26A1J10_SC	LO	MOD_REFER	(A42-10 DTOA_POWER LO)
26A1J10_SC	LO	POWER	(A26_CABINET_PWR VOLTS BAD)
26A1J9_SC	HI	MOD_REFER	(A42-5 DTOA_INHIBIT HI)
26A1J9_SC	LO	MOD_REFER	(A42-5 DTOA_INHIBIT LO)
A1A10J1	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A10J1	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A10J1	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A10J1	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A10J1	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A10J1	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A10J2	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A10J2	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A10J2	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A10J2	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A10J2	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A10J2	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A10J2	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A10J2	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A10J2	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A10J2	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A10J2	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A10J2	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A10J2	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A10J2	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A10J2	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A10J2	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A10J2	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A10J2	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A10J2	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A10J2	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A10J2	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A10J2	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A10J2	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A10J2	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A10J2	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A10J2	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A10J2	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A10J2	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A10J2	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)

A1A10J4	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A10J4	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A10J4	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A10J4	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A10J4	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A10J4	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A10J4	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A10J4	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A10J4	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A10J4	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A10J4	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A10J4	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A10J4	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A10J4	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A10J4	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A10J4	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A10J4	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A10J4	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A10J4	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A10J4	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A10J4	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A10J4	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A10J4	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A10J4	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A10J4	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A10J4	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A10J4	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A10J4	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A10J4	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A10J5	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A10J5	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A10J5	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A10J5	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A10J5	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A10J5	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A10J6	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A10J6	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A10J6	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A10J6	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A10J7	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A10J7	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A10J7	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A10J7	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A10J8	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A10J8	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A10J8	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A10J8	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A10J8	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A10J8	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A10J9	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A10J9	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A10J9	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A10J9	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)

A1A10J9	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A10J9	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A10J9	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A10J9	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A10J9	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A10J9	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A10J9	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A10J9	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A10J9	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A10J9	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A10J9	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A10J9	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A10J9	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A10J9	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A10J9	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A10J9	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A10J9	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A10J9	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A10J9	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A10J9	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A10J9	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A10J9	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A10J9	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A10J9	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A10J9	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A11J14	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J14	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J14	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J14	BAD	*TIMING*	(A15 FUNCTIONS BAD)
A1A11J15	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J15	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J15	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J15	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A11J18	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J18	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J18	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J18	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A11J2	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J2	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J2	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J3	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J3	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J3	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J4	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J4	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J4	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J5	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J5	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J5	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J6	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)

A1A11J6	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J6	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J7	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J7	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J7	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J8	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J8	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J8	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A11J9	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A11J9	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A11J9	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A13J1_F	LO	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A13J1_F	LO	TIMING	(A13J1 WAVEFORM BAD)
A1A13J1_F	HI	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A13J1_F	HI	TIMING	(A13J1 WAVEFORM BAD)
A1A13J1_L	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A13J1_L	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A1J1	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A1J1	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A1J1	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A1J1	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A1J4	BAD	A1_DELAY_LINE	(A1J4 LOGIC_LEVELS BAD)
A1A1J4	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A1J5	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A1J5	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A1J5	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A1J5	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A1J6	BAD	A1_DELAY_LINE	(A1-9 ST3 BAD)
A1A1J6	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A1J6	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A1J6	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A1J7	BAD	A2_DELAY_LINE	(A2-22 NOT_CL2 BAD)
A1A1J7	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A1J7	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A1J7	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A1J8	BAD	A1_DELAY_LINE	(A1-9 ST3 BAD)
A1A1J8	BAD	A3_DELAY_LINE	(A3-9 DST3 BAD)
A1A1J8	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A1J8	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A1J8	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A1J9	BAD	A1_DELAY_LINE	(A1J9 LOGIC_LEVELS BAD)
A1A1J9	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A20J11	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A20J11	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)

A1A20J11	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A20J11	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A20J11	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A20J11	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A20J11	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A20J11	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A20J11	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A20J11	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A20J11	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A20J11	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A20J11	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A20J11	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A20J11	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A20J11	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A20J11	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A20J11	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A20J11	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A20J11	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A20J11	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A20J11	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A20J11	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A20J11	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A20J11	BAD	PMFL	(A18 CONTROL_FUNCTIONS BAD)
A1A20J11	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A20J11	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A20J11	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A20J11	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A20J11	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A20J13	FAULTED	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A20J13	FAULTED	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A20J13	FAULTED	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A20J13	FAULTED	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A20J13	FAULTED	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A20J13	FAULTED	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A20J13	FAULTED	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A20J13	FAULTED	A17_AMPL_CONTROL	(A17 AMPL_CORREL_LEFT_BEAMS BAD)
A1A20J13	FAULTED	A22_SIGN_CONTROL	(A22 GATES_OR_PWR_OR_MODULE BAD)
A1A20J13	FAULTED	A22_SIGN_CONTROL	(A7J4 LOGIC_LEVELS BAD)
A1A20J13	FAULTED	A22_SIGN_CONTROL	(A7J4 SIGN_LEFT_BEAMS BAD)
A1A20J13	FAULTED	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A20J13	FAULTED	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A20J13	FAULTED	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A20J13	FAULTED	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A20J13	FAULTED	A7_DELAY_LINE	(A7-9 ST1 BAD)
A1A20J13	FAULTED	A7_DELAY_LINE	(A7-21 NOT_CL1 BAD)
A1A20J13	FAULTED	A7_DELAY_LINE	(A7J9 LOGIC_LEVELS BAD)
A1A20J13	FAULTED	A7_DELAY_LINE	(A7J4 LOGIC_LEVELS BAD)
A1A20J13	FAULTED	A7_DELAY_LINE	(A7-13 SIGN_LEFT_BEAMS BAD)
A1A20J13	FAULTED	A9_DELAY_LINE	(A9-9 DST1 BAD)
A1A20J13	FAULTED	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A20J13	FAULTED	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A20J13	FAULTED	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A20J13	FAULTED	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A20J13	FAULTED	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A20J13	FAULTED	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A20J13	FAULTED	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A20J13	FAULTED	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)

A1A20J13	FAULTED	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A20J13	FAULTED	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A20J13	FAULTED	MUX_REF	(A70J7 FREQ BAD)
A1A20J13	FAULTED	MUX_REF	(A70J7 VOLTS LO)
A1A20J13	FAULTED	MUX_REF	(A70J7 VOLTS HI)
A1A20J13	FAULTED	PMFL	(A18 TEST_FUNCTIONS BAD)
A1A20J13	FAULTED	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A20J13	FAULTED	TIMING	(A13J1 WAVEFORM BAD)
A1A20J13	FAULTED	TIMING	(A14 FUNCTIONS BAD)
A1A20J13	FAULTED	TIMING	(A15 FUNCTIONS BAD)
A1A20J13	FAULTED	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A20J2	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A20J2	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A20J2	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A20J2	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A20J2	BAD	TIMING	(A20J2 GATE_1 BAD)
A1A20J3	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A20J3	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A20J3	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A20J3	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A20J3	BAD	TIMING	(A20J3 GATE_2 BAD)
A1A20J4	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A20J4	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A20J4	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A20J4	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A20J4	BAD	TIMING	(A20J4 GATE_3 BAD)
A1A20J5	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A20J5	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A20J5	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A20J5	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A20J5	BAD	TIMING	(A20J5 GATE_4 BAD)
A1A20J9	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A20J9	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A20J9	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A20J9	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A20J9	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A20J9	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A20J9	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A20J9	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A20J9	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A20J9	BAD	A29_DELAY_LINE	A29-21 NOT_ST1 BAD)
A1A20J9	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A20J9	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A20J9	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A20J9	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A20J9	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A20J9	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A20J9	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A20J9	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A20J9	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A20J9	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A20J9	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A20J9	BAD	MUX_REF	(A70J7 FREQ BAD)

A1A20J9	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A20J9	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A20J9	BAD	PMFL	(A18 CONTROL_FUNCTIONS BAD)
A1A20J9	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A20J9	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A20J9	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A20J9	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A20J9	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A21J11	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A21J11	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A21J11	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A21J11	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A21J11	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A21J11	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A21J13	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A21J13	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A21J13	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A21J13	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A21J13	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A21J13	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A21J13	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A21J13	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A21J13	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A21J13	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A21J13	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A21J13	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A21J13	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A21J13	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A21J13	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A21J13	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A21J13	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A21J13	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A21J13	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A21J13	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A21J13	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A21J13	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A21J13	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A21J13	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A21J13	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A21J13	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A21J13	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A21J13	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A21J13	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A21J14	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A21J14	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A21J14	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A21J14	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A21J14	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A21J14	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A21J14	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A21J14	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A21J14	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A21J14	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A21J14	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A21J14	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)

A1A21J14	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A21J14	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A21J14	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A21J14	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A21J14	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A21J14	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A21J14	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A21J14	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A21J14	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A21J14	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A21J14	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A21J14	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A21J14	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A21J14	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A21J14	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A21J14	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A21J14	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A21J8	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A21J8	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A21J8	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A21J8	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A21J8	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A21J8	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A21J8	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A21J8	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A21J8	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A21J8	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A21J8	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A21J8	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A21J8	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A21J8	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A21J8	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A21J8	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A21J8	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A21J8	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A21J8	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A21J8	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A21J8	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A21J8	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A21J8	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A21J8	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A21J8	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A21J8	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A21J8	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A21J8	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A21J8	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A24J4	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A24J4	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A24J4	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A24J4	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A24J4	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A24J4	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A24J4	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A24J4	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A24J4	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A24J4	BAD	MUX_REF	(A70J7 FREQ BAD)

A1A24J4	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A24J4	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A24J4	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A24J5	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A24J5	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A24J5	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A24J5	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A24J5	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A24J5	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A24J5	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A24J5	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A24J5	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A24J5	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A24J5	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A24J5	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A24J5	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A24J5	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A24J5	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A29J6	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A29J6	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A29J6	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A29J6	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A29J7	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A29J7	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A29J7	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A29J7	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A29J8	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A29J8	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A29J8	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A29J8	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A29J8	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A29J8	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A2J6	BAD	A1_DELAY_LINE	(A1-9 ST3 BAD)
A1A2J6	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A2J6	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A2J6	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A30J1	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A30J1	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A30J1	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A30J1	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A30J1	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A30J1	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A30J1	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A30J1	BAD	A22_SIGN_CONTROL	(A22 GATES_OR_PWR_OR_MODULE BAD)
A1A30J1	BAD	A22_SIGN_CONTROL	(A7J4 LOGIC_LEVELS BAD)
A1A30J1	BAD	A22_SIGN_CONTROL	(A7J4 SIGN_LEFT_BEAMS BAD)
A1A30J1	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A30J1	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A30J1	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A30J1	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A30J1	BAD	A7_DELAY_LINE	(A7-9 ST1 BAD)
A1A30J1	BAD	A7_DELAY_LINE	(A7-21 NOT_CL1 BAD)

A1A30J1	BAD	A7_DELAY_LINE	(A7J9 LOGIC_LEVELS BAD)
A1A30J1	BAD	A7_DELAY_LINE	(A7J4 LOGIC_LEVELS BAD)
A1A30J1	BAD	A7_DELAY_LINE	(A7-13 SIGN_LEFT_BEAMS BAD)
A1A30J1	BAD	A9_DELAY_LINE	(A9-9 DST1 BAD)
A1A30J1	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A30J1	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A30J1	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A30J1	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A30J1	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A30J1	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A30J1	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A30J1	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A30J1	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A30J1	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A30J1	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A30J1	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A30J1	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A30J1	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A30J1	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A30J1	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A30J1	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A30J1	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A30J2	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A30J2	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A30J2	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A30J2	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A30J2	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A30J2	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A30J2	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A30J2	BAD	A17_AMPL_CONTROL	(A17 AMPL_CORREL_LEFT_BEAMS BAD)
A1A30J2	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A30J2	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A30J2	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A30J2	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A30J2	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A30J2	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A30J2	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A30J2	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A30J2	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A30J2	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A30J2	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A30J2	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A30J2	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A30J2	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A30J2	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A30J2	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A30J2	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A30J2	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A30J2	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A30J2	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A30J2	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A30J2	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A30J3	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A30J3	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A30J3	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A30J3	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)

A1A30J3	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A30J3	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A30J3	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A30J3	BAD	A17_AMPL_CONTROL	(A17 AMPL_CORREL_LEFT_BEAMS BAD)
A1A30J3	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A30J3	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A30J3	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A30J3	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A30J3	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A30J3	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A30J3	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A30J3	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A30J3	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A30J3	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A30J3	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A30J3	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A30J3	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A30J3	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A30J3	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A30J3	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A30J3	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A30J3	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A30J3	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A30J3	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A30J3	BAD	TIMING	(A15 FUNCTIONS BAD)
A1A30J3	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A30J4_NEG	LO	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A30J4_NEG	LO	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A30J4_NEG	LO	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A30J4_NEG	LO	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A30J4_NEG	LO	A17_AMPL_CONTROL	(A17 AMPL_CORREL_LEFT_BEAMS BAD)
A1A30J4_NEG	LO	A17_AMPL_CONTROL	(A17-OUT AMPL_LEFT_BEAMS BAD)
A1A30J4_NEG	LO	A1_DELAY_LINE	(A1-9 ST3 BAD)
A1A30J4_NEG	LO	A1_DELAY_LINE	(A1J9 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A1_DELAY_LINE	(A1J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A1_DELAY_LINE	(A1-13 BIT_0_LEFT_BEAMS BAD)
A1A30J4_NEG	LO	A22_SIGN_CONTROL	(A22 GATES_OR_PWR_OR_MODULE BAD)
A1A30J4_NEG	LO	A22_SIGN_CONTROL	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A22_SIGN_CONTROL	(A7J4 SIGN_LEFT_BEAMS BAD)
A1A30J4_NEG	LO	A23_ATOD	(A23 AMPLITUDE BAD)
A1A30J4_NEG	LO	A23_ATOD	(A23-6 SIGN BAD)
A1A30J4_NEG	LO	A23_ATOD	(A23-3 INVERTED_SIGN BAD)
A1A30J4_NEG	LO	A24_AMP	(A24 FUNCTIONS BAD)
A1A30J4_NEG	LO	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A30J4_NEG	LO	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A30J4_NEG	LO	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A30J4_NEG	LO	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A30J4_NEG	LO	A2_DELAY_LINE	(A2-22 NOT_CL2 BAD)
A1A30J4_NEG	LO	A30_DTOA	(A30 FUNCTIONS BAD)
A1A30J4_NEG	LO	A3_DELAY_LINE	(A3-9 DST3 BAD)
A1A30J4_NEG	LO	A4_DELAY_LINE	(A4-9 ST2 BAD)
A1A30J4_NEG	LO	A4_DELAY_LINE	(A4-21 NOT_CL2 BAD)
A1A30J4_NEG	LO	A4_DELAY_LINE	(A4J9 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A4_DELAY_LINE	(A4J4 LOGIC_LEVELS BAD)

A1A30J4_NEG	LO	A4_DELAY_LINE	(A4-13 BIT_1_LEFT_BEAMS BAD)
A1A30J4_NEG	LO	A5_DELAY_LINE	(A5-9 DST2 BAD)
A1A30J4_NEG	LO	A7_DELAY_LINE	(A7-9 ST1 BAD)
A1A30J4_NEG	LO	A7_DELAY_LINE	(A7-21 NOT_CL1 BAD)
A1A30J4_NEG	LO	A7_DELAY_LINE	(A7J9 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A7_DELAY_LINE	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	LO	A7_DELAY_LINE	(A7-13 SIGN_LEFT_BEAMS BAD)
A1A30J4_NEG	LO	A9_DELAY_LINE	(A9-9 DST1 BAD)
A1A30J4_NEG	LO	CORRELATOR_REF	(A24 FUNCTIONS BAD)
A1A30J4_NEG	LO	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A30J4_NEG	LO	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A30J4_NEG	LO	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A30J4_NEG	LO	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A30J4_NEG	LO	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A30J4_NEG	LO	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A30J4_NEG	LO	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A30J4_NEG	LO	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A30J4_NEG	LO	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A30J4_NEG	LO	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A30J4_NEG	LO	MOD_REFER	(A42-5 DTOA_INHIBIT HI)
A1A30J4_NEG	LO	MOD_REFER	(A42-5 DTOA_INHIBIT LO)
A1A30J4_NEG	LO	MOD_REFER	(A42-10 DTOA_POWER LO)
A1A30J4_NEG	LO	MUX_REF	(A70J7 FREQ BAD)
A1A30J4_NEG	LO	MUX_REF	(A70J7 VOLTS LO)
A1A30J4_NEG	LO	MUX_REF	(A70J7 VOLTS HI)
A1A30J4_NEG	LO	PMFL	(A18 CONTROL_FUNCTIONS BAD)
A1A30J4_NEG	LO	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A30J4_NEG	LO	TIMING	(A20J2 GATE_1 BAD)
A1A30J4_NEG	LO	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A30J4_NEG	LO	TIMING	(A13J1 WAVEFORM BAD)
A1A30J4_NEG	LO	TIMING	(A14 FUNCTIONS BAD)
A1A30J4_NEG	LO	TIMING	(A15 FUNCTIONS BAD)
A1A30J4_NEG	HI	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A30J4_NEG	HI	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A30J4_NEG	HI	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A30J4_NEG	HI	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A30J4_NEG	HI	A17_AMPL_CONTROL	(A17 AMPL_CORREL_LEFT_BEAMS BAD)
A1A30J4_NEG	HI	A17_AMPL_CONTROL	(A17-OUT AMPL_LEFT_BEAMS BAD)
A1A30J4_NEG	HI	A1_DELAY_LINE	(A1-9 ST3 BAD)
A1A30J4_NEG	HI	A1_DELAY_LINE	(A1J9 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A1_DELAY_LINE	(A1J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A1_DELAY_LINE	(A1-13 BIT_0_LEFT_BEAMS BAD)
A1A30J4_NEG	HI	A22_SIGN_CONTROL	(A22 GATES_OR_PWR_OR_MODULE BAD)
A1A30J4_NEG	HI	A22_SIGN_CONTROL	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A22_SIGN_CONTROL	(A7J4 SIGN_LEFT_BEAMS BAD)
A1A30J4_NEG	HI	A23_ATOD	(A23 AMPLITUDE BAD)
A1A30J4_NEG	HI	A23_ATOD	(A23-6 SIGN BAD)
A1A30J4_NEG	HI	A23_ATOD	(A23-3 INVERTED_SIGN BAD)
A1A30J4_NEG	HI	A24_AMP	(A24 FUNCTIONS BAD)
A1A30J4_NEG	HI	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A30J4_NEG	HI	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A30J4_NEG	HI	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A30J4_NEG	HI	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A30J4_NEG	HI	A2_DELAY_LINE	(A2-22 NOT_CL2 BAD)
A1A30J4_NEG	HI	A30_DTOA	(A30 FUNCTIONS BAD)

A1A30J4_NEG	HI	A3_DELAY_LINE	(A3-9 DST3 BAD)
A1A30J4_NEG	HI	A4_DELAY_LINE	(A4-9 ST2 BAD)
A1A30J4_NEG	HI	A4_DELAY_LINE	(A4-21 NOT_CL2 BAD)
A1A30J4_NEG	HI	A4_DELAY_LINE	(A4J9 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A4_DELAY_LINE	(A4J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A4_DELAY_LINE	(A4-13 BIT_1_LEFT_BEAMS BAD)
A1A30J4_NEG	HI	A5_DELAY_LINE	(A5-9 DST2 BAD)
A1A30J4_NEG	HI	A7_DELAY_LINE	(A7-9 ST1 BAD)
A1A30J4_NEG	HI	A7_DELAY_LINE	(A7-21 NOT_CL1 BAD)
A1A30J4_NEG	HI	A7_DELAY_LINE	(A7J9 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A7_DELAY_LINE	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_NEG	HI	A7_DELAY_LINE	(A7-13 SIGN_LEFT_BEAMS BAD)
A1A30J4_NEG	HI	A9_DELAY_LINE	(A9-9 DST1 BAD)
A1A30J4_NEG	HI	CORRELATOR_REF	(A24 FUNCTIONS BAD)
A1A30J4_NEG	HI	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A30J4_NEG	HI	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A30J4_NEG	HI	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A30J4_NEG	HI	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A30J4_NEG	HI	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A30J4_NEG	HI	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A30J4_NEG	HI	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A30J4_NEG	HI	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A30J4_NEG	HI	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A30J4_NEG	HI	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A30J4_NEG	HI	MOD_REFER	(A42-5 DTOA_INHIBIT HI)
A1A30J4_NEG	HI	MOD_REFER	(A42-5 DTOA_INHIBIT LO)
A1A30J4_NEG	HI	MOD_REFER	(A42-10 DTOA_POWER HI)
A1A30J4_NEG	HI	MUX_REF	(A70J7 FREQ BAD)
A1A30J4_NEG	HI	MUX_REF	(A70J7 VOLTS LO)
A1A30J4_NEG	HI	MUX_REF	(A70J7 VOLTS HI)
A1A30J4_NEG	HI	PMFL	(A18 CONTROL_FUNCTIONS BAD)
A1A30J4_NEG	HI	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A30J4_NEG	HI	TIMING	(A20J2 GATE_1 BAD)
A1A30J4_NEG	HI	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A30J4_NEG	HI	TIMING	(A13J1 WAVEFORM BAD)
A1A30J4_NEG	HI	TIMING	(A14 FUNCTIONS BAD)
A1A30J4_NEG	HI	TIMING	(A15 FUNCTIONS BAD)
A1A30J4_POS	LO	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A30J4_POS	LO	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A30J4_POS	LO	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A30J4_POS	LO	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A30J4_POS	LO	A17_AMPL_CONTROL	(A17 AMPL_CORREL_LEFT_BEAMS BAD)
A1A30J4_POS	LO	A17_AMPL_CONTROL	(A17-OUT AMPL_LEFT_BEAMS BAD)
A1A30J4_POS	LO	A1_DELAY_LINE	(A1-9 ST3 BAD)
A1A30J4_POS	LO	A1_DELAY_LINE	(A1J9 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A1_DELAY_LINE	(A1J4 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A1_DELAY_LINE	(A1-13 BIT_0_LEFT_BEAMS BAD)
A1A30J4_POS	LO	A22_SIGN_CONTROL	(A22 GATES_OR_PWR_OR_MODULE BAD)
A1A30J4_POS	LO	A22_SIGN_CONTROL	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A22_SIGN_CONTROL	(A7J4 SIGN_LEFT_BEAMS BAD)
A1A30J4_POS	LO	A23_ATOD	(A23 AMPLITUDE BAD)
A1A30J4_POS	LO	A23_ATOD	(A23-6 SIGN BAD)
A1A30J4_POS	LO	A23_ATOD	(A23-3 INVERTED_SIGN BAD)
A1A30J4_POS	LO	A24_AMP	(A24 FUNCTIONS BAD)

A1A30J4_POS	LO	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A30J4_POS	LO	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A30J4_POS	LO	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A30J4_POS	LO	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A30J4_POS	LO	A2_DELAY_LINE	(A2-22 NOT_CL2 BAD)
A1A30J4_POS	LO	A30_DTOA	(A30 FUNCTIONS BAD)
A1A30J4_POS	LO	A3_DELAY_LINE	(A3-9 DST3 BAD)
A1A30J4_POS	LO	A4_DELAY_LINE	(A4-9 ST2 BAD)
A1A30J4_POS	LO	A4_DELAY_LINE	(A4-21 NOT_CL2 BAD)
A1A30J4_POS	LO	A4_DELAY_LINE	(A4J9 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A4_DELAY_LINE	(A4J4 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A4_DELAY_LINE	(A4-13 BIT_1_LEFT_BEAMS BAD)
A1A30J4_POS	LO	A5_DELAY_LINE	(A5-9 DST2 BAD)
A1A30J4_POS	LO	A7_DELAY_LINE	(A7-9 ST1 BAD)
A1A30J4_POS	LO	A7_DELAY_LINE	(A7-21 NOT_CL1 BAD)
A1A30J4_POS	LO	A7_DELAY_LINE	(A7J9 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A7_DELAY_LINE	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_POS	LO	A7_DELAY_LINE	(A7-13 SIGN_LEFT_BEAMS BAD)
A1A30J4_POS	LO	A9_DELAY_LINE	(A9-9 DST1 BAD)
A1A30J4_POS	LO	CORRELATOR_REF	(A24 FUNCTIONS BAD)
A1A30J4_POS	LO	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A30J4_POS	LO	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A30J4_POS	LO	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A30J4_POS	LO	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A30J4_POS	LO	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A30J4_POS	LO	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A30J4_POS	LO	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A30J4_POS	LO	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A30J4_POS	LO	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A30J4_POS	LO	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A30J4_POS	LO	MOD_REFER	(A42-5 DTOA_INHIBIT HI)
A1A30J4_POS	LO	MOD_REFER	(A42-5 DTOA_INHIBIT LO)
A1A30J4_POS	LO	MOD_REFER	(A42-10 DTOA_POWER LO)
A1A30J4_POS	LO	MUX_REF	(A70J7 FREQ BAD)
A1A30J4_POS	LO	MUX_REF	(A70J7 VOLTS LO)
A1A30J4_POS	LO	MUX_REF	(A70J7 VOLTS HI)
A1A30J4_POS	LO	PMFL	(A18 CONTROL_FUNCTIONS BAD)
A1A30J4_POS	LO	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A30J4_POS	LO	TIMING	(A20J2 GATE_1 BAD)
A1A30J4_POS	LO	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A30J4_POS	LO	TIMING	(A13J1 WAVEFORM BAD)
A1A30J4_POS	LO	TIMING	(A14 FUNCTIONS BAD)
A1A30J4_POS	LO	TIMING	(A15 FUNCTIONS BAD)
A1A30J4_POS	HI	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A30J4_POS	HI	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A30J4_POS	HI	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A30J4_POS	HI	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A30J4_POS	HI	A17_AMPL_CONTROL	(A17 AMPL_CORREL_LEFT_BEAMS BAD)
A1A30J4_POS	HI	A17_AMPL_CONTROL	(A17-OUT AMPL_LEFT_BEAMS BAD)
A1A30J4_POS	HI	A1_DELAY_LINE	(A1-9 ST3 BAD)
A1A30J4_POS	HI	A1_DELAY_LINE	(A1J9 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A1_DELAY_LINE	(A1J4 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A1_DELAY_LINE	(A1-13 BIT_0_LEFT_BEAMS BAD)
A1A30J4_POS	HI	A22_SIGN_CONTROL	(A22 GATES_OR_PWR_OR_MODULE BAD)
A1A30J4_POS	HI	A22_SIGN_CONTROL	(A7J4 LOGIC_LEVELS BAD)

A1A30J4_POS	HI	A22_SIGN_CONTROL	(A7J4 SIGN_LEFT_BEAMS BAD)
A1A30J4_POS	HI	A23_ATOD	(A23 AMPLITUDE BAD)
A1A30J4_POS	HI	A23_ATOD	(A23-6 SIGN BAD)
A1A30J4_POS	HI	A23_ATOD	(A23-3 INVERTED_SIGN BAD)
A1A30J4_POS	HI	A24_AMP	(A24 FUNCTIONS BAD)
A1A30J4_POS	HI	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A30J4_POS	HI	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A30J4_POS	HI	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A30J4_POS	HI	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A30J4_POS	HI	A2_DELAY_LINE	(A2-22 NOT_CL2 BAD)
A1A30J4_POS	HI	A30_DTOA	(A30 FUNCTIONS BAD)
A1A30J4_POS	HI	A3_DELAY_LINE	(A3-9 DST3 BAD)
A1A30J4_POS	HI	A4_DELAY_LINE	(A4-9 ST2 BAD)
A1A30J4_POS	HI	A4_DELAY_LINE	(A4-21 NOT_CL2 BAD)
A1A30J4_POS	HI	A4_DELAY_LINE	(A4J9 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A4_DELAY_LINE	(A4J4 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A4_DELAY_LINE	(A4-13 BIT_1_LEFT_BEAMS BAD)
A1A30J4_POS	HI	A5_DELAY_LINE	(A5-9 DST2 BAD)
A1A30J4_POS	HI	A7_DELAY_LINE	(A7-9 ST1 BAD)
A1A30J4_POS	HI	A7_DELAY_LINE	(A7-21 NOT_CL1 BAD)
A1A30J4_POS	HI	A7_DELAY_LINE	(A7J9 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A7_DELAY_LINE	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_POS	HI	A7_DELAY_LINE	(A7-13 SIGN_LEFT_BEAMS BAD)
A1A30J4_POS	HI	A9_DELAY_LINE	(A9-9 DST1 BAD)
A1A30J4_POS	HI	CORRELATOR_REF	(A24 FUNCTIONS BAD)
A1A30J4_POS	HI	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A30J4_POS	HI	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A30J4_POS	HI	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A30J4_POS	HI	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A30J4_POS	HI	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A30J4_POS	HI	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A30J4_POS	HI	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A30J4_POS	HI	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A30J4_POS	HI	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A30J4_POS	HI	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A30J4_POS	HI	MOD_REFER	(A42-5 DTOA_INHIBIT HI)
A1A30J4_POS	HI	MOD_REFER	(A42-5 DTOA_INHIBIT LO)
A1A30J4_POS	HI	MOD_REFER	(A42-10 DTOA_POWER HI)
A1A30J4_POS	HI	MUX_REF	(A70J7 FREQ BAD)
A1A30J4_POS	HI	MUX_REF	(A70J7 VOLTS LO)
A1A30J4_POS	HI	MUX_REF	(A70J7 VOLTS HI)
A1A30J4_POS	HI	PMFL	(A18 CONTROL_FUNCTIONS BAD)
A1A30J4_POS	HI	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A30J4_POS	HI	TIMING	(A20J2 GATE_1 BAD)
A1A30J4_POS	HI	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A30J4_POS	HI	TIMING	(A13J1 WAVEFORM BAD)
A1A30J4_POS	HI	TIMING	(A14 FUNCTIONS BAD)
A1A30J4_POS	HI	TIMING	(A15 FUNCTIONS BAD)
A1A30J4_WAV	BAD	A10_DELAY_LINE	(A10J9 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A10_DELAY_LINE	(A10J2 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A10_DELAY_LINE	(A10-21 NOT_CL1 BAD)
A1A30J4_WAV	BAD	A10_DELAY_LINE	(A10-9 ST1 BAD)
A1A30J4_WAV	BAD	A10_DELAY_LINE	(A10J4 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A10_DELAY_LINE	(A10J9 RE1 BAD)
A1A30J4_WAV	BAD	A10_DELAY_LINE	(A10-10 NOT_RE1 BAD)
A1A30J4_WAV	BAD	A17_AMPL_CONTROL	(A17 AMPL_CORREL_LEFT_BEAMS BAD)
A1A30J4_WAV	BAD	A17_AMPL_CONTROL	(A17-OUT AMPL_LEFT_BEAMS BAD)

A1A30J4_WAV	BAD	A1_DELAY_LINE	(A1-9 ST3 BAD)
A1A30J4_WAV	BAD	A1_DELAY_LINE	(A1J9 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A1_DELAY_LINE	(A1J4 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A1_DELAY_LINE	(A1-13 BIT_0_LEFT_BEAMS BAD)
A1A30J4_WAV	BAD	A22_SIGN_CONTROL	(A22 GATES_OR_PWR_OR_MODULE BAD)
A1A30J4_WAV	BAD	A22_SIGN_CONTROL	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A22_SIGN_CONTROL	(A7J4 SIGN_LEFT_BEAMS BAD)
A1A30J4_WAV	BAD	A23_ATOD	(A23 AMPLITUDE BAD)
A1A30J4_WAV	BAD	A23_ATOD	(A23-6 SIGN BAD)
A1A30J4_WAV	BAD	A23_ATOD	(A23-3 INVERTED_SIGN BAD)
A1A30J4_WAV	BAD	A24_AMP	(A24J5 DELTIC_REF BAD)
A1A30J4_WAV	BAD	A24_AMP	(A24 FUNCTIONS BAD)
A1A30J4_WAV	BAD	A26_REFERENCE_CONTROL	(A26 FUNCTIONS BAD)
A1A30J4_WAV	BAD	A29_DELAY_LINE	(A29-21 NOT_ST1 BAD)
A1A30J4_WAV	BAD	A29_DELAY_LINE	(A29-9 DST1 BAD)
A1A30J4_WAV	BAD	A2_DELAY_LINE	(A2-22 NOT_CL2 BAD)
A1A30J4_WAV	BAD	A30_DTOA	(A30 FUNCTIONS BAD)
A1A30J4_WAV	BAD	A3_DELAY_LINE	(A3-9 DST3 BAD)
A1A30J4_WAV	BAD	A4_DELAY_LINE	(A4-9 ST2 BAD)
A1A30J4_WAV	BAD	A4_DELAY_LINE	(A4-21 NOT_CL2 BAD)
A1A30J4_WAV	BAD	A4_DELAY_LINE	(A4J9 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A4_DELAY_LINE	(A4J4 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A4_DELAY_LINE	(A4-13 BIT_1_LEFT_BEAMS BAD)
A1A30J4_WAV	BAD	A5_DELAY_LINE	(A5-9 DST2 BAD)
A1A30J4_WAV	BAD	A7_DELAY_LINE	(A7-9 ST1 BAD)
A1A30J4_WAV	BAD	A7_DELAY_LINE	(A7-21 NOT_CL1 BAD)
A1A30J4_WAV	BAD	A7_DELAY_LINE	(A7J9 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A7_DELAY_LINE	(A7J4 LOGIC_LEVELS BAD)
A1A30J4_WAV	BAD	A7_DELAY_LINE	(A7-13 SIGN_LEFT_BEAMS BAD)
A1A30J4_WAV	BAD	A9_DELAY_LINE	(A9-9 DST1 BAD)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A70J5 FREQ BAD)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A70J7 FREQ BAD)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A70J5 VOLTS LO)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A70J7 VOLTS LO)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A70J5 VOLTS HI)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A70J7 VOLTS HI)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A43J1 DELTIC_REF BAD)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A43J3 DELTIC_REF BAD)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A24J4 DELTIC_REF BAD)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A24J5 DELTIC_REF BAD)
A1A30J4_WAV	BAD	CORRELATOR_REF	(A24 FUNCTIONS BAD)
A1A30J4_WAV	BAD	MOD_REFER	(A42-5 DTOA_INHIBIT HI)
A1A30J4_WAV	BAD	MOD_REFER	(A42-5 DTOA_INHIBIT LO)
A1A30J4_WAV	BAD	MUX_REF	(A70J7 FREQ BAD)
A1A30J4_WAV	BAD	MUX_REF	(A70J7 VOLTS LO)
A1A30J4_WAV	BAD	MUX_REF	(A70J7 VOLTS HI)
A1A30J4_WAV	BAD	PMFL	(A18 CONTROL_FUNCTIONS BAD)
A1A30J4_WAV	BAD	POWER	(A26_CABINET_PWR VOLTS BAD)
A1A30J4_WAV	BAD	TIMING	(A20J2 GATE_1 BAD)
A1A30J4_WAV	BAD	TIMING	(A21J9 END_BEAM_STORE BAD)
A1A30J4_WAV	BAD	TIMING	(A13J1 WAVEFORM BAD)
A1A30J4_WAV	BAD	TIMING	(A14 FUNCTIONS BAD)
A1A30J4_WAV	BAD	TIMING	(A15 FUNCTIONS BAD)